Preliminary Definition of CORTEX Programming Model

P. Barron, G. Biegel, V. Cahill, A. Casimiro, S. Clarke,
R. Cunningham, A. Fitzpatrick, G. Gaertner, B. Hughes,
J. Kaiser, R. Meier and P. Veríssimo

DI-FCUL TR–03–15

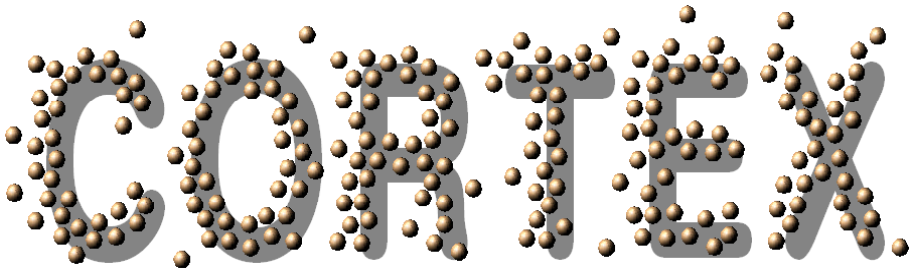
July 2003

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1700 Lisboa
Portugal

Technical reports are available at http://www.di.fc.ul.pt/tech-reports. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Project IST-2000-26031

CO-operating Real-time senTient objects:
architecture and EXperimental evaluation



Preliminary Definition of CORTEX Programming Model

CORTEX Deliverable D2

Version 1.0

March 31, 2002

Revisions

Rev.	Date	Comment
1.0	31/03/2002	Integrated document

Editor

René Meier, Trinity College Dublin

Contributors

René Meier, Trinity College Dublin
Greg Biegel, Trinity College Dublin
Aidan Fitzpatrick, Trinity College Dublin
Barbara Hughes, Trinity College Dublin
Raymond Cunningham, Trinity College Dublin
António Casimiro Costa, University of Lisbon
Jörg Kaiser, University of Ulm
Vinny Cahill, Trinity College Dublin
Peter Barron, Trinity College Dublin
Gregor Gaertner, Trinity College Dublin
Siobhán Clarke, Trinity College Dublin
Paulo Veríssimo, University of Lisbon

Address

Department of Computer Science,
Trinity College Dublin,
Ireland

Table of Contents

Chapter 1: Overview

As described in [1], the objective of work package one is to design a programming model suitable for the development of proactive applications constructed from mobile sentient objects. Associated tasks include the definition of the concept of hierarchically structured sentient objects in a language independent way, the definition of primitives made available to the application developer to control and manage sentient objects as well as the definition of primitives for inter-object communication based on the event paradigm, and the definition of a mechanism for Quality of Service (QoS) specification based on the use of application-level parameters that can be mapped onto the system level QoS parameters characterizing the service levels that can be supported by the underlying physical infrastructure at a given point in time.

This document represents deliverable D2 of the CORTEX project providing the preliminary definition of the CORTEX programming model. It embodies an early deliverable that will be followed in the second half of the project by its successor deliverable D6, the final definition of the CORTEX programming model.

The preliminary definition of the CORTEX programming model presented in this document is structured as a collection of technical reports, some of which have been submitted for publication. This document is divided into four major parts, each addressing one of the objectives of work package one outlined above. Each part consists of one or more technical reports discussing the corresponding objective.

The remainder of this chapter introduces the four major parts into which this document is divided in detail. Chapter 2: outlines our definition of sentient objects and Chapter 3: presents how an application developer may program them in a context aware manner. The event-based communication model made available to the application programmer to define inter-object communication is described in Chapter 4: . And finally, Chapter 5: presents a first approach to specifying QoS parameters characterizing the level of service supported by the underlying infrastructure.

1.1 Sentient Object Model

The initial description of the CORTEX project [1] envisages the construction of applications out of proactive, mobile, context-aware entities that it terms sentient objects. Applications will be composed of large numbers of sentient objects, which will interact with each other and with the environment according to the specifics of the application logic. Various application scenarios that describe how this will happen are presented in CORTEX deliverable D1 [2].

As sentient objects are central to the CORTEX project, the first point of investigation for the CORTEX programming model is to form a clearer understanding of what sentient objects are; what their key properties are and how they will function. In Chapter 2: of this document, we present our initial ideas on sentient objects. We distinguish sentient objects, sensors and actuators, and present a complete categorization of all entities in CORTEX in terms of their abilities to interact with each other and with the external (real-world) environment. We present an initial definition for sensors, actuators and sentient objects based on these categorizations. In addition, we describe our initial ideas on how these entities will come together to form applications such as those described in [2], and we address some of the issues arising from our specification. Finally, we outline what we see as the next step in our investigation into sentient objects. We identify major challenges and suggest potential approaches to address them appropriately within the context of CORTEX.

1.1.1 Smart Sensors as Sentient Objects

The notion of sentient objects is central to CORTEX. It is seen as a paradigm, which allows to model applications, which interact with their physical environment. One of the basic building blocks which corresponds to a sentient object may be a smart sensor or actuator. When directly interacting with the physical environment, this component is also called a smart transducer [3]. The notion of a smart transducer as a basic networked building block recently became popular as indicated by a request for proposal from OMG [4]. The characteristics emphasize the ability to co-operate in an object-oriented decentralized setting. A smart transducer has many characteristics, which map well to the notion of a sentient object, which gives a much higher and more general level of abstraction. As Alireza Moini [5] points out: "smart sensors are information sensors, not transducers and signal processing elements". This points to their active role in a system, their ability for self-organization, i.e. spontaneous interaction with other smart components, and their autonomy of behaviour.

The Intelligent Camera Unit (ICU) is an example for a sentient component, which has been developed for autonomous mobile robots as a smart optical sensor for line tracking. A real-time executive suited for ICU and the publisher/subscriber communication software has partly been developed in the context of CORTEX. Rather than transferring the pixels of the raw image, ICU includes a microcontroller for detecting the colour, position and the slope of a tracking line. This information is directly used by the smart motor controller of a mobile robot to adjust the speed of the left and right motors accordingly to follow the line. As described in the paper [6], the synchronization between the CMOS image sensor and the microcontroller requires tight temporal control. However, this low level timing is completely hidden from the outside. The interface is defined by an input and an output channel. The input channel takes configuration messages to start and stop the periodic dissemination of the position information and to define the rate and priority at which this information is published.

1.2 Context Awareness

CORTEX defines the environment of sentient objects as constituting an interaction and communication channel and being in the control and awareness loop of the objects [1]. This is significant, in that sentient objects are not constrained to communication through traditional network channels, but may use 'hidden' channels in the environment. Such hidden channels often provide a faster communication mechanism than traditional network channels, which is important when considering the real time nature of sentient objects in CORTEX.

In order to utilize the environment as a channel for interaction and communication, sentient objects need to have an awareness of the environment in which they operate. Context awareness is analogous to environmental awareness and provides a sentient object with information sensed from the environment, which may be used in interactions with other sentient objects and / or the fulfillment of its goals.

Sentient objects use sensors to sense information from their environment and actuators to make changes to the environment and in this way may achieve communication through the environment.

To make sentient objects context aware, three components have been identified which need to be incorporated into sentient objects. A capturing function abstracts raw sensor data into a more useful form, a model of the world is maintained through the representation of past, current and future context, while an inference engine causes the behavior of sentient objects to influenced by their context.

1.3 Event-Based Communication Model

As described in [1], an event-based communication model supporting anonymous one-to-many communication is well suited to provide dynamic pattern of communication arising from the unpredictable manner in which the potentially mobile sentient objects that comprise

an application interact between themselves and the environment. These unpredictable interaction pattern may depend not only on the common interest shared by communicating peers, but also on the geographical proximity of sentient object to one another and the environment.

Chapter 4: presents two technical reports addressing the requirements of event-based programming models. The first report describes the programming model of event-based middleware designed to provide inter-object communication pattern for mobile objects in a wireless environment utilizing an ad-hoc network model. It outlines an approach for event-based communication where objects communicate based on events of a certain type, which represent the common interest shared by a group of interacting objects, as well as based on their geographical location. The notion of proximity is introduced to allow an application programmer to define a geographical area within which a particular event type is propagated, thus providing a means to define the scope within which specific events are valid. Furthermore, a novel approach to allow objects to express interest, or lack thereof, in events of a certain type or containing certain parameter values is outlined. This novel approach to filtering of events also supports the ability of a system to easily cope with dynamically joining and leaving objects as well as its ability to grow.

The second report presents a survey of existing event systems structured as a taxonomy of distributed event-based programming systems. It identifies a set of fundamental properties of event-based programming systems and categorizes them according to the event model and event service criteria. The taxonomy serves as a means to identify the properties of event-based middleware in the CORTEX context and is used as basis to discuss the design of the event-based programming model proposed in the first report.

1.4 QoS Specification

The programming model needs to take into account the provision of incremental real-time and reliability guarantees. To achieve this, the development of a means to express quality of service (QoS) properties in the programming model, where QoS is a metric of predictability in terms of timeliness and reliability, is required. In addition, a global model for QoS assurance across heterogeneous physical networks, must be developed. This dual QoS requirement will be exposed to the programming model via a high-level abstraction of the underlying internetwork, coupled with the novel idea of a hierarchy of zones of guaranteed levels of QoS.

An abstract network model, enabling the specification of application level QoS parameters, coupled with a mapping mechanism from these abstract requirements, to system level properties, will be provided to the programming model. Thus, specific QoS guarantees per network, whilst maintaining a complete separation of application developer from the heterogeneity of the physical network, will be achievable.

The basis for the CORTEX architecture is to model the underlying communication infrastructure hierarchically, structured as a WAN-of-CAN. The QoS available for the internetwork varies per network. Individual networks can be viewed as guaranteeing a given QoS degree. For example, strong timeliness guarantees for CANs and weaker guarantees for wireless networks. To take advantage of the varying timing guarantees, each QoS area will be visible as a hierarchy of zones, with each zone capable of delivering specific levels of QoS. A group may be completely contained within a zone, or may span many zones. In the latter case, the QoS must adapt to the weakest QoS guaranteed for the weakest zone. Thus, adaptability, coupled with predictability must be available via this hierarchy of zones approach.

The high-level abstract network model, the mappings from application-level to system properties and the hierarchy of zones for QoS containment will be further discussed in Chapter 5: of this document.

Chapter 2: Sentient Object Model

This chapter is divided into to two major sections. Section 2.1 presents a technical report introducing a description of the properties and operations of sentient objects and section 2.2 presents a paper on smart sensors describing a potential example of a sentient object that has been presented at the IEEE International Conference on Mechatronics and Machine Vision in Practice [6], which took place in August 2001.

2.1 Sentient Objects

The CORTEX annex [COR01] describes sentient objects as intelligent, mobile, context-aware software components. It envisages applications composed of a great many of these objects, acting autonomously to achieve the application goals. Where necessary, objects will interact to ensure successful completion of goals or to improve system performance. However, the annex fails to supply anything more than high-level notions of what is meant by a sentient object. Consequently, it was felt that the first goal of the project should be to try and more precisely define both the properties and operation of sentient objects.

2.1.1 Initial Ideas

Examination of the sample CORTEX application scenarios [COR02] was the main source of inspiration for initial ideas on sentient objects. Figure (1) below shows a possible sentient object model suggested by these scenarios.

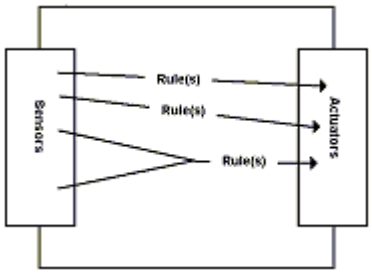


Figure (1): Early sentient object model

The sentient object is an encapsulated entity, with its interfaces being sensors and actuators. The actuators are controlled according to sensor input, via a rule-based system. For example, considering the ATTS scenarios as defined in [COR02], a light sensor might be connected to a headlight actuator via a rule that states that the headlight should be turned on if it is dark. However, one can envisage the rule structure becoming extremely complex in larger sentient objects, where data from multiple sensors must be considered simultaneously, and decisions may require complex behaviour from multiple actuators. In addition, this model is obviously very loosely specified, and does not answer many questions about sentient objects. Some of the major issues that we identified were:

- What is the granularity of a sentient object?
- What do we mean by sensors and actuators in this context – are they hardware devices, or software abstractions thereof?
- How do sentient objects interact?
- What sorts of hierarchy/relationships can exist between sentient objects, if any?

In addition to answering the questions above, one of the main challenges to be overcome in defining a sentient object was to constrain our definition. The most obvious definition of a sentient object was as something that senses and actuates. However, this is hugely broad classification, which can easily encompass a great many existing computer systems. For example, a desktop computer could be said to sense user input via the mouse or keyboard and actuate on its environment by moving a cursor or displaying a character on the monitor. A more precise definition was needed, which would allow us to state more categorically whether or not a given entity was a sentient object.

2.1.2 Classification

In CORTEX, we identify two distinct categories of events: software events and real-world events. Software events are the main form of interaction between entities in CORTEX and provide anonymous, ad-hoc communication. Real-world events are anything that happens in the environment, either causing a change of state in a sensor or caused by an actuator. We propose that, using these two categories, we can identity and distinguish the different entities that may exist in CORTEX. This is to be done by categorizing any encapsulated entity in terms of the classes of event it can consume, and the classes it can produce. Figure (2) below depicts an entity that both consumes and produces all categories of event. Another entity might only be able to produce and consume software events, for example. The different possibilities for production and consumption lead to the different classifications.

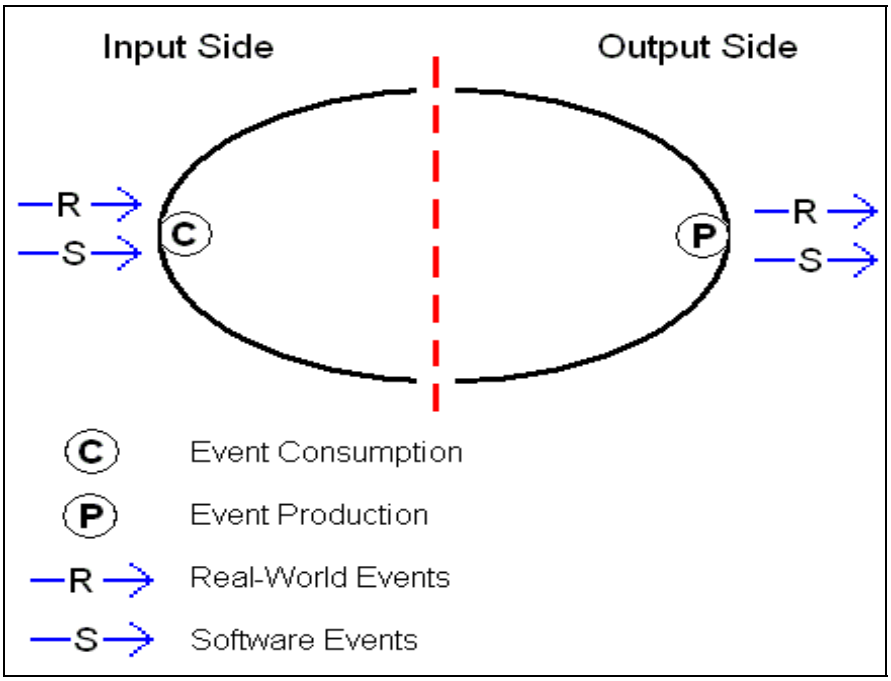


Figure (2): The production and consumption of events

Our initial investigation of the different possibilities for production and consumption considered that an entity could produce only one category of event and consume only one category of event. The resulting classification yields four distinct entity types, which we categorized as follows:

- Real-world consumption, software production

- This class of object produces software events in response to real world events. This seems to suggest that these entities are sensors.

- Software consumption, software production
  - This class of entity both consumes and produces software events. One can envisage that application logic will be distributed across entities of this class. This implies that these are sentient objects.

- Software consumption, real world production
  - These entities produce real world events in response to the consumption of software events. They have the opposite consumption/production properties to the entities that we have identified as sensors, and hence would appear to be actuators.

- Real-world consumption, real-world production
  - This class of entity produces real-world events in response to real-world events consumed. This implies that it is a simply real-world system; in the context of CORTEX, we propose to term such entities sentient systems.

From these classifications, we are able to form initial definitions for the entities in CORTEX:

"A sensor is an entity that produces software events in reaction to a stimulus detected by some real-world hardware device"

"An actuator is an entity which consumes software events, and reacts by attempting to change the state of the real world in some way via some hardware device."

"A sentient object is an entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator."

Our original definition of a sentient object was simply an entity that both produced and consumed software events. However, this definition would have included all objects programmed on current event systems. In order to constrain the definition further, the idea of a control path between sensor(s) and actuator(s) was specified. By doing this, we limit the type of entity in which we are interested to those with real-world interactions. One possibility for a further constraint to be imposed may be that sentient objects should be required to respond in a timely manner, but as yet this has not been specified.

2.1.3 System Operation

The classifications are useful in that they distinguish between hardware entities, software objects, and the intermediate devices. The idea of a sentient system maps quite well to the notions in [COR01] and [COR02], where, within the various application scenarios, cars, airplanes, etc are identified as the mobile, sentient objects. In our view, these entities would be considered to be sentient systems, and their on-board devices that allow both observation of and interaction with the environment would be sensors and actuators, respectively. Internal objects that consume software events and produce corresponding software events where appropriate would be components of the application logic, known as sentient objects. A depiction of a sentient object is shown in figure (3) below.

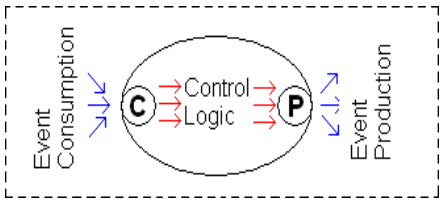


Figure (3): A sentient object

The application logic could be separated across as few or as many of these objects as the application programmer deems necessary. This will hopefully give the benefits of object-oriented programming, i.e. re-usability of code modules, separation of logic, etc. Also, we hope this will allow sentient objects to be composed of other, smaller sentient objects in a hierarchal manner. Whether other properties of object oriented software, such as polymorphism and inheritance, can be applied to sentient objects remains to be investigated.

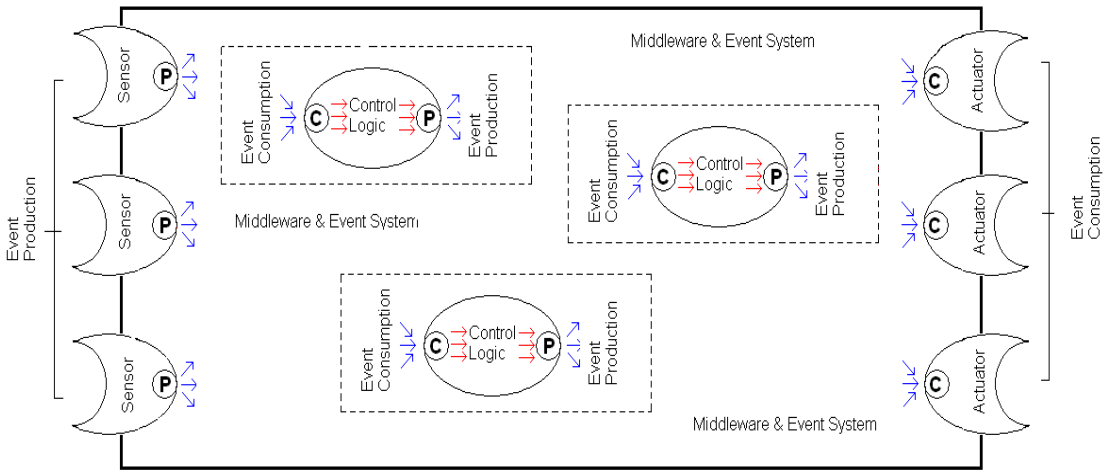


Figure (4): Envisaged operation of a CORTEX application

Figure (4) above shows the envisaged operation of a CORTEX-type system. The boundaries of this entity would most likely be equivalent to the boundaries of a CAN (Controller Area Network) as in the WAN-of-CAN architecture [COR01]. So, the entire system depicted above might represent a car in the ATTS scenario or an aircraft in the ATC scenario [COR02]. While this may provide a useful abstraction to the application programmer, situations can be envisaged where sentient systems, or even sentient objects, may span multiple hardware devices across the WAN[1]. For the sake of simplicity, we may later specify that this is not allowed, or perhaps give entities spanning multiple hardware devices different names. However, according to the current definitions, entities existing across multiple hardware devices are permissible.

2.1.4 Classification II

In the classifications above, we only consider entities that can produce one category of event and can consume one category of event. However, we must also consider the case where an

[1] The latter case is only likely to be possible with complex sentient objects, which are composed from more primitive objects.

entity is capable of producing and/or consuming events in more than one domain. For example, consider the case illustrated in figure (5) below:

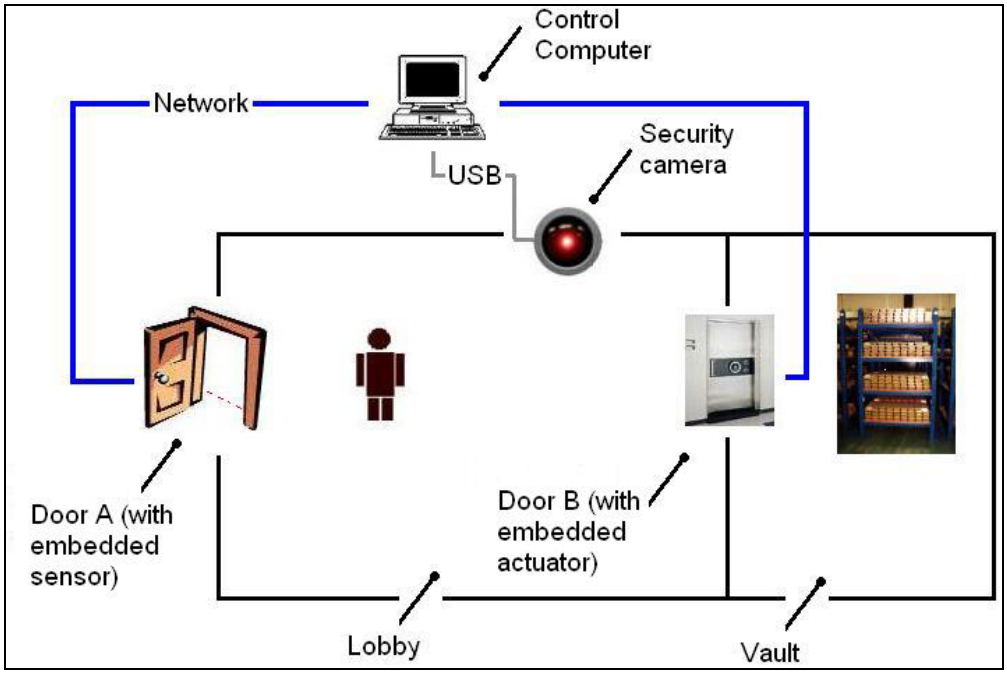


Figure (5): Example of consumption from two domains

The scene depicted is a possible security system for a bank vault. The frame of Door A is fitted with an embedded sensor system (e.g. infra-red) that detects people passing through the door. This sensor is connected, via a CORTEX sensor wrapper object, to a network. The detection of an individual entering the lobby (real-world event) causes a software event to be published on the network. The CORTEX application, which is running on the control computer, is subscribed to this type of event. When such an event is delivered, the application retrieves an image from the security camera, which is connected to the computer via a USB cable, and is communicating with the application in a non-CORTEX domain. If the person is recognized, an event is issued to cause Door B to open, admitting them to the vault. The non-CORTEX communication is crucial, as this means that the application running on the control computer is itself providing the CORTEX sensor wrapper for the camera. Hence, it is receiving both software events from Door A's embedded systems, and real-world events from the camera. This dual consumption is not well described by the classifications above.

The possibility of new scenarios such as that described above increases the potential number of entity classifications to nine. We look at each of new classifications in turn, and consider their usefulness and feasibility. Firstly, we look at two new entity classifications that appear to be an enhancement on our original definitions for sensors and actuators, as they allow for such scenarios.

- Real-world and software consumption, software production
    - The scenario above describes an entity of this class. It can be viewed as a sensor that also accepts software events as input.
- Software consumption, real-world and software production

- This entity classification can be seen an actuator that also generates software events as output.

The three remaining entity classifications seem to be of less value, as no scenarios have been conceived in which they are useful or necessary. They all deal with entities that can both produce and consume real-world events. In this respect, they can all be seen as enhancements to the definition of a sentient system, allowing for software consumption and production as well. For clarity, we present the three remaining classifications below.

- Real-world and software consumption, real-world and software production
- Real-world and software consumption, real-world production
- Real-world consumption, real-world and software production

In light of these previously unconsidered entity classifications, we must necessarily reconsider our definitions for sensors, actuators and sentient object. In classifying these new entity types, we suggest that real-world events are in some way *"more defining"* than software events. By this, we mean that the consumption and/or production of real-world events are more crucial to the classification of an entity in CORTEX than the consumption/production of software events. This means that if an entity can consume real-world events, then it will be classified as a sensor, regardless of whether or not it can consume software events. Similarly, if an entity can produce real-world events, then it will be considered to be an actuator. Entities that neither produce nor consume real-world events are sentient objects[2]. From this, we have our new, revised, definitions of the entities in CORTEX:

"A sentient object is a entity that both consumes and produces software events, and lies in some control path between at least one sensor and one actuator." (Unchanged)

"A sensor is a entity that produces software events in reaction to a stimulus detected by some real-world hardware device(s). A sensor may also (optionally) receive software events as input."

"A actuator is an entity which consumes software events and responds by attempting to change the state of the real world via some hardware device(s). An actuator may also (optionally) produce software events as output"

2.1.5 Related Ideas and Technologies

An area of interest in the initial investigation into sentient objects was the distinction between what we were trying to achieve with sentient objects, and the work of the artificial intelligence community in the field of agents. A review of agent literature revealed a large number of different definitions of what an agent was. Some of these were particularly interesting from a CORTEX viewpoint. The two most relevant definitions are quoted below.

[2] These assertions assume the all entities are capable of consuming at least one category of event, and are capable of producing at least one category of event. In addition, they do not apply to entities that both consume and produce real-world events – those entities that we have previously termed 'sentient systems'.

The AIMA agent [Russell and Norvig 1995]: *"An agent is anything which can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"*

The Brustoloni Agent [Brustoloni 1991]: *"Autonomous agents are systems capable of autonomous, purposeful action in the real world. Agents must be reactive; that is, be able to respond to external, asynchronous stimuli in a timely fashion."*

While the definitions presented above appear in many ways to be quite similar to the descriptions of sentient objects found in the CORTEX annex [COR01], they represent only a philosophy of artificial intelligence programming, as opposed to the more rigid application development framework that CORTEX hopes to present. However, further examination of agent-based programming may well become useful at a later stage in the project, when we more fully consider how sentient objects will be programmed.

Another topic that may become more useful as we progress with our investigation into sentient objects is that of visual programming. From what we have already defined about sentient objects, and from what we hope to accomplish, a system for creating applications composed of these objects can be envisaged which uses a graphical user interface rather than more traditional coding methods. A visual model is particularly suited to the application domain in which we are interested, and it is hoped that this will allow the programmer to create applications in a quicker and more intuitive fashion. Altaira [PFE01] is a simple example of the use of a visual programming system in the development of context-aware applications. It is a visual language for the control of mobile robots. It uses icons to represent sensors and actuators, and employs a rule-based logic to specify the behavior of the robot.

The context toolkit [DEY01] is an additional existing technology that may be of use to use in defining how we create and program sentient objects. It draws on the ideas of graphical toolkits and widget libraries in order to provide the application programmer with reusable building blocks for applications that use environment/context awareness. In the CORTEX programming model, we wish to make context information available to the application with minimal additional work for the programmer. It is hoped the work and ideas of the context toolkit can be expanded upon to provide context information to sentient objects in a simple, re-useable manner.

2.1.6 Future Work

Obviously, we are still far from having a complete definition of what a sentient object is, and of how sentient objects will be programmed. We must next consider issues of hierarchy and composition for sentient objects, and examine in more detail the relationships between the entities that we have identified so far – how these entities will come together to form larger systems, how larger entities can be composed from more primitive ones, etc. In addition, we will consider the interactions that will take place between different entities, via the real world and also via the event-based communication service, which we must also specify in more detail. The main challenge in this is to select an event model that is capable of providing suitable abstractions of the facilities that the lower levels of the CORTEX model aims to provide, such as group communication and quantifiable QoS, and is also capable of the anonymous, ad-hoc communication that CORTEX applications will require. Finally, we must examine issues of context – how the different entities in CORTEX will perceive and interact with the environment and how they with represent their surroundings internally. We hope to provide mechanisms that will simplify the use of context at an application level, shielding the programmer from the complexities of obtaining and manipulating contextual information. These mechanisms will probably be similar to those seen in the context toolkit. From here, we hope to be able to more clearly specify the programming model for CORTEX.

2.1.7 References

[COR01] CORTEX – "Annex 1, Description of Work", October 2000.

[COR02] CORTEX – "Definition of Application Scenarios", October 2001.

[HAA01] M. Haahr, R Meier, P. Nixon, V. Cahill, E. Jul – "Filtering and Scalability in the ECO Distributed Event Model", April 2000.

[DEY01] A. Dey, G. Abowd – "Towards a better understanding of Context and Context-Awareness", September 1999

[PFE01] J. Pfeiffer – "A Rule-Based Visual Language for Small Mobile Robots", September 1997

2.2 ICU - A Smart Optical Sensor for Direct Robot Control

Jörg Kaiser and Peter Schaeffer

Department of Computer Structures, University of Ulm, Germany

e-mail: kaiser@informatik.uni-ulm.de, ps1@informatik.uni-ulm.de

Abstract-- Technological advances will allow the integration of smart devices which may comprise sensor components, computational devices and a network interface. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables. The paper deals with two issues. Firstly, it describes an architecture which encourages the design of multi-level control hierarchies exploiting the easy availability of application related sensor information. This is based on encapsulated smart components, a well defined communication interface and the easy access to this information by a variant of a shared data space. Secondly, we describe the functions and the hardware of ICU which constitutes the prototype of an optical sensor designed for vehicle guidance operating as a smart component in such a system. The task of the sensor is to detect a guidance line and directly produce the information needed by the steering system to control the vehicle.

Index terms-- Vehicle guidance, co-operative control, low cost sensor, smart sensor, middleware, CAN-Bus interface

2.2.1 Introduction

Technological advances will allow the integration of smart devices which may comprise sensor components, computational devices and a network interface. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables**.** Consequently, the interfaces and the functions of these smart components may include functions related to overall control, supervision, and maintenance issues. In his excellent survey of vision sensors, Alireza Moini [1] makes the point that "smart sensors are information sensors, not transducers and signal processing elements". Perhaps the most interesting and challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture in which smart autonomous components co-operate to control physical processes reactively without the need of a central co-ordination facility. In such a system, multiple different sensors will co-operate to augment perception of the environment and autonomous actuators will co-ordinate actions to increase speed, power and quality of actuation thus forming decentralized virtual sensor and actuator networks.

As an example consider the vision tasks of an autonomous mobile robot. There may be multiple levels of image processing and recognition ranging from line tracking over obstacle avoidance to scene recognition and image understanding [2]. All these tasks need different levels of reactivity. With multiple dedicated low cost/low power sensors, the need for a fast reactive behaviour can be met without interfering with higher levels. The output of the tracking sensor can directly be used by the motor drive system to keep the vehicle on a guidance line, the obstacle detection sensor may directly stop the motors with minimal

latency. Higher level analysis may use the same sources of data but usually is less predictable and responsive because it will have to relate multiple sensors in more complex planning and recognition tasks.

In the context of a project dealing with adequate models and middleware techniques for communication and co-ordination in control systems [3], we developed ICU (Intelligent Camera Unit). ICU is an optical sensor which computes and disseminates information which is directly related to actuator control rather than deliver just the raw image for further processing.

The rest of the paper is organized as follows. Section 2 motivates a distributed control architecture composed from smart components. In 3, we describe the requirements and intentions of such an architecture and briefly sketch the main ideas. The functions and basics of ICU are covered in 4 while section 5 details the hardware architecture. Conclusions and acknowledgements are given in 6 and 7, respectively.

2.2.2 Co-operating Optical Sensors

There is a large variety of optical sensors and vision sensors ranging from very simple CCD and CMOS arrays to intelligent vision chips and artificial retinas [1]. They can detect relatively simple things as distributions of light intensities or detection of marks or lines. More sophisticated sensors enable motion detection and deriving the speed of a device from optical flow analysis directly embedded in the sensor array [4]. These devices are usually specialized to a single function which they can perform faster, better and with less energy as compared to traditional vision systems which usually convert a digital image in a an analogue format coming from the first days of television and then use a frame grabber easily wasting 30 Mbyte of PCI bandwidth again to create a digital image in the processor memory. The advantage of this approach is its generality, i.e. that an image, once in memory is the raw material which can be analysed by software in any desired aspect. However, high performance processors are needed resulting in an overall complex and rather power consuming system. For autonomous mobile vehicles this may be a problem. On the other hand, specialized optical devices only perform a single task. Therefore, multiple such devices may be needed to collaboratively sense and improve the perception of the environment. Special purpose sensors can operate in different spectral ranges and can be used to built robust systems with the possibility to use adaptive strategies to replace defective functions by “virtual sensors” combining the virtues of active sensors, probably with lower precision, timeliness or resolution. To combine these sensors in an effective and robust way and enable direct actuator control, distributed middleware is needed to support easy diffusion of the information to each entity which may be interested in the data, to support spontaneous generation of information and allow temporal constraints to be specified on information propagation. Moreover, it is desirable to have a low configuration effort when integrating new sensors.

2.2.3 Requirements for the System Control Architecture

Traditional control systems usually center around a single control unit (CU) which has a sophisticated real world interface. The CU is the last link in a chain of transducers, converting a physical process variable, like a temperature, a pressure or an optical signal via a electrical signal to a digital representation. Signal conditioning as shaping analogue values, debouncing digital inputs and improving an image received from a camera all are performed by this CU. The resulting digital information is a raw, conditioned representation of a single physical entity. Subsequent processing, fusion of multiple sensors information, and generation of control signals for the actuators also is performed by the same processor. Because all these tasks have widely differing performance requirements and temporal constraints, complex planning and scheduling schemes have to guarantee that these tasks can be properly

coordinated in the temporal domain. Simply moving to multiprocessors usually worsens the scheduling problem [5].

Using decentralized smart sensors and actuators puts computing power to the place where it is needed. These components are autonomous systems which perform a dedicated complex task beyond signal conditioning. Interconnected by a communication network, they encapsulate a certain functionality and provide meaningful application related information at their network interface. Kopetz [6] highlights the importance of an adequate interface to support functional and temporal encapsulation of smart components to support the composability of an application.

There are a number of goals which we want to reach for the control system architecture:

1. Components of the network are autonomous. Autonomy means that each component is in its own sphere of control and no control signal should cross the boundary of a component. Hence, components only interact via shared information as e.g. proposed in the data field architecture in ADS (Autonomous Decentralized Systems)[7] or the tuple space in Linda [8]. As a consequence, interfaces can be created which do not rely on any, possibly time critical control transfers. This supports easy extensibility, reliability and robustness and encourages a component-based design.
2. The architecture should support many-to-many communication patterns. A typical situation is that the information gained from a sensor can be used and analyzed in more than one place, e.g., the output of a our ICU optical sensor on a mobile robot is interesting for reactive motor control implemented on a small micro-controller as well as for long term navigation strategies implemented on a more powerful device. Another typical example is the situation in which control commands issued from a controller address a number of identical actuators; e.g., all motors have to stop in case of emergency.
3. Communication is spontaneous because control systems have to react to external events. These external events are recognized at the sensor interface of an embedded system at arbitrary points in time and lead to communication activities to disseminate the information. This is best captured in a generative, event-based communication model [9, 10, 11].
4. Communication is anonymous. Consider again the example of stopping a set of motors. When issuing a stop command, it is not of interest to address a specific motor, rather it must be ensured that all relevant motors receive the command. Similarly, when reacting to a stop command, it is not of interest which controller has issued that command. On a more abstract level, a sensor object triggered by the progression of time or the occurrence of an event spontaneously generates the respective information and distributes it to the system. Thus, it is considered as a producer. The corresponding consumer objects have mechanisms to determine whether this information is useful for them. This interaction leads to a model of anonymous communication in which the producer does not know which consumers will use its information and, vice versa, the consumers only know which information they need independently from which source they receive it. Furthermore, anonymous communication supports the extensibility and the reliability of the system because objects can be added or be replaced easily without changing address information maintained in the other objects.

To meet these requirements, we adopted a publisher/subscriber model of communication in which producers (publishers) and consumers (subscribers) of data are connected via event channels [7, 8]. In contrast to other forms of a shared information space [9, 10], the semantic of event channels integrates the notification of consumers when an event occurs and thus support the temporal relation of the event occurrence and the notification of the subscribers. Event channels support content-based communication by relating an event channel to a certain class of information rather than to a source or destination of a message. Thus, a message is routed by its content which is dynamically bound to a network related address. A

more detailed description of this architecture can be found in [3]. What mainly distinguishes our publisher/subscriber scheme from existing schemes like the event service in Jini[11] is its anonymity and compared to the event service in CORBA[13] the distributed implementation.

In our system, an event channel is related to a certain class of information, like e.g. certain classes of tracking information. Publishers may be smart cameras, infrared sensors and alike all pushing tracking information in the respective event channels. Multiple subscribers e.g. motor control or the navigation system receive this information and can locally filter and use it. The publisher/subscriber architecture was particularly designed to enable interoperation of embedded systems which usually have stringent performance and bandwidth constraints with more powerful networks and processors. At the moment we provide interoperability between a CAN-Bus [14] for low level reactive control and IP-based protocols. This allows a robot in a co-operating team to seamlessly access remote sensor information of another robot e.g. via a wireless IP connection.

Autonomous components encourage the design of layered intelligent systems as they were proposed by [2], [15], [16]. Most authors separate the reactive layer from higher system levels which realize deliberative behavior. Reactive behavior maps to the low level control architecture in which resources have to be provided to support fast and timely behavior with respect to safety critical actions like avoiding obstacles, recognizing landmarks or coordinating actions of multiple actuator devices. Deliberative behaviors include to a large extend activities related to planning like map building, path planning, global navigation and action co-ordination. Reactive and deliberative behaviors may be based on the same set of sensor information, however with a different degree of relating sensor information and different temporal attributes. This even may include co-ordination of different robots each equipped with a set of different sensors co-operating to jointly explore an unknown environment.

A prototype of our publisher/subscriber architecture is implemented under Linux and RT-Linux. Linux handles the non-critical communication over IP while RT-Linux is used for the time critical communication on the CAN-Bus. A gateway connects the two networks [17]. The testbed is a KURT II autonomous robot which besides various distance and odometric sensors is equipped with ICU (Fig.1).

Fig.1 Kurt-2 robots, robot in front carries ICU (courtesy of Michael Mock, Institute of Autonomous intelligent Systems (AiS))

2.2.4 Functions of the ICU

ICU was primarily designed for vehicle guidance in an experimental traffic scenario. In our first application ICU tracks a guidance line (G-line) on the floor which may include embedded coloured marks. ICU disseminates the position and orientation of the G-line and the colour of the embedded marks. The information delivered by ICU is then directly used by the motor drive system to reactively keep the vehicle to follow this line, to slow down or to stop if a certain coloured mark is detected. Fig. 2 shows the principles of detecting a G-line (the white bar in the figure). The orientation of the G-line is calculated from three scan lines (S-lines). Because of the special colour representation (see Fig. 4) of the image sensor an S-line is represented by 4 pixel-lines.

ICU tries to detect the white G-line on a darker background. To do this, the image is transferred to the micro-controller which calculates the colour and brightness values of every pixel on the S-line. The maximal value of brightness is then used to detect the G-line. Therefore, the G-line must be a white line with optionally embedded coloured marks (see Fig. 2 and 3).

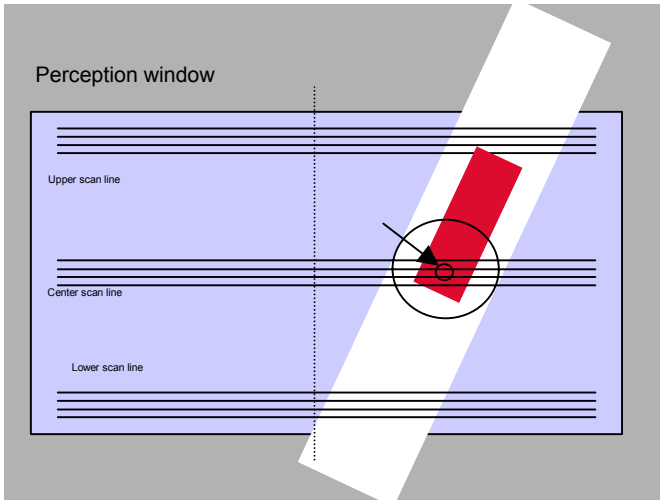


Fig.2 Principle of line detection

The problem encountered in determining the overall brightness is the colour representation. The vision sensor uses a Bayer scheme [18] as depicted in Fig. 4. Hence, the colour of an individual pixel has to be inferred from the relative brightness of the adjacent pixels. Having determined the colour of a pixel, its brightness is calculated using the weighted colour values as described below. The Bayer colour encoding uses 4 pixels to represent the values of red, green and blue.

ICU tries to detect the white G-line on a darker background. To do this, the image is transferred to the micro-controller which calculates the colour and brightness values of every pixel on the S-line. The maximal value of brightness is then used to detect the G-line. Therefore, the G-line must be a white line with optionally embedded coloured marks (see Fig. 2 and 3).

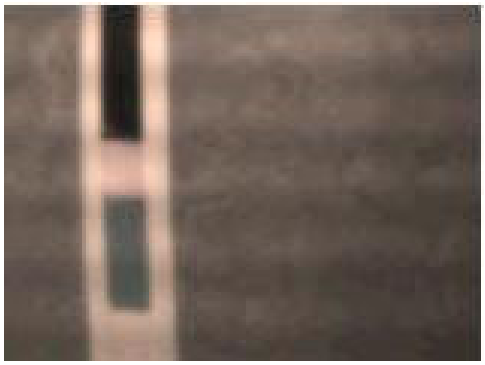


Fig.3 A line as ICU sees it

The problem encountered in determining the overall brightness is the colour representation.

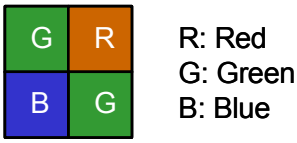


Fig. 4 Bayer Scheme of colouring

The vision sensor uses a Bayer scheme [18] as depicted in Fig. 4. Hence, the colour of an individual pixel has to be inferred from the relative brightness of the adjacent pixels. Having determined the colour of a pixel, its brightness is calculated using the weighted colour values as described below. The Bayer colour encoding uses 4 pixels to represent the values of red, green and blue. Therefore, taking the straightforward approach to directly use these values to represent a single coloured pixel would decrease the resolution of the image by a factor of 4 which was not acceptable. Hence, we used a standard colour encoding scheme which exploits the brightness values of a 3x3 neighbourhood as depicted in Fig. 5. Fig. 5 shows how the colour of a (single) pixel is calculated. Even though the spectral sensitivity of a single pixel is confined to a certain colour, an RGB-value will be assigned to it by considering the adjacent pixels. The RGB-value is calculated according to the weights of the adjacent pixels as depicted in Fig. 5. Subsequently, the brightness of every pixel is calculated on the basis of the RGB-values by the equation:

$$\text{Brightness } B = (R*38+G*75+B*15)/128$$

Thus we maintain the full resolution of the image in spite of the Bayer colour representation, however, the algorithm has the effect of the low pass filter. As a consequence, the image inevitably looses sharpness.

The next step is detecting the G-line. For this, the S-line is scanned from both sides to find the respective transition of contrast. The algorithm is rather straightforward for the moment and only checks whether the edges detected from the left-to-right and right-to-left scan have a certain distance from each other corresponding to the assumed width of the G-line. If not, a fault value is returned. All three S-lines are evaluated as depicted in Fig. 2 to determine the position and orientation of the tracking line.

The last step is to determine the colour of the marks embedded in the G-line. As Fig. 3 shows it calculates the position of the centre of the G-line and takes the colour and the brightness of the pixel at this position. If the G-line is not detected properly, ICU interpolates the respective centre position from the positions of the G-line in the upper and the lower S-line. Subsequently, the colour of the pixel at this interpolated position is selected. This approach is also useful for intersections or junctions of G-lines.

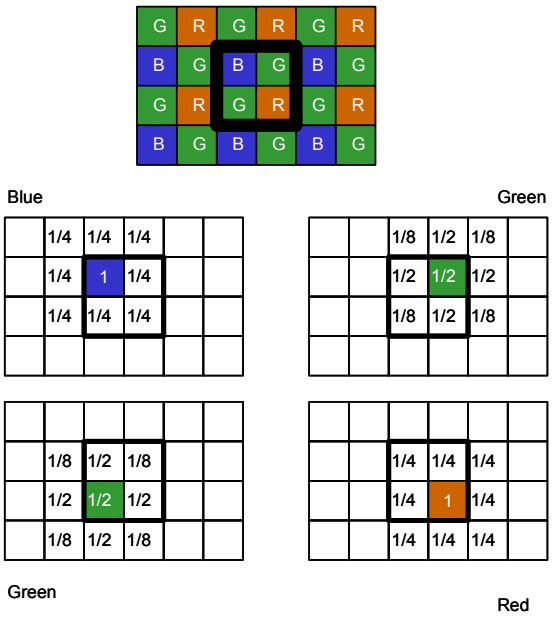


Fig. 5 Colour encoding

The sensor is rather sensitive to the precise focus of the lens. Currently, because the sensor always has a fixed distance to the G-line, focussing has only been performed once and therefore is done by hand. Because ICU usually does not provide an image for inspection to adjust the focus, we provided a focussing aid based on an automatic detection of a maximum contrast transition of an appropriate black to white edge. We only use green pixels for the mechanism because colour is not needed and green pixels are dense on the sensor (see Fig. 4 and 5) whereas lines of blue or red pixels always exhibit gaps. The green pixels are arranged in a zick-zack pattern which, however, does not have a major influence on the detection of a black-to-white transistion.

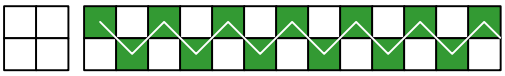


Fig. 5 Scanning for focussing control

If the lens is out of focus, the contrast edge is blurred meaning that the values of adjacent pixel do not exhibit a sharp transition. By adjusting the focus precisely, adjacent pixels indicate a sharp transition which in turn triggers an LED for external inspection. The condition for the LED is a maximal brightness difference that is above a certain threshold between any two adjacent pixels.

2.2.5 The Hardware of ICU

ICU was intended as a low cost sensor for simple imaging tasks which can directly by used for control applications. ICU should be adaptable to various simple vision tasks. Therefore, we took a camera/processor approach and put emphasis on an easy and fairly universal processor interface. Fig. 6 depicts the basic components of ICU. We connect the sensor to an asynchronous external port, which is not the most efficient way but eases the use of different micro-controllers. In fact, we used the front end sensor and the interface logic even with a simple micro-controller as Motorola 68HC11 for educational purposes. In the current version, a 16-Bit controller C167 from Siemens [19] (now provided by Infineon) is used featuring a minimal instruction execution time of 100ns and a peak rate of 10Mips.

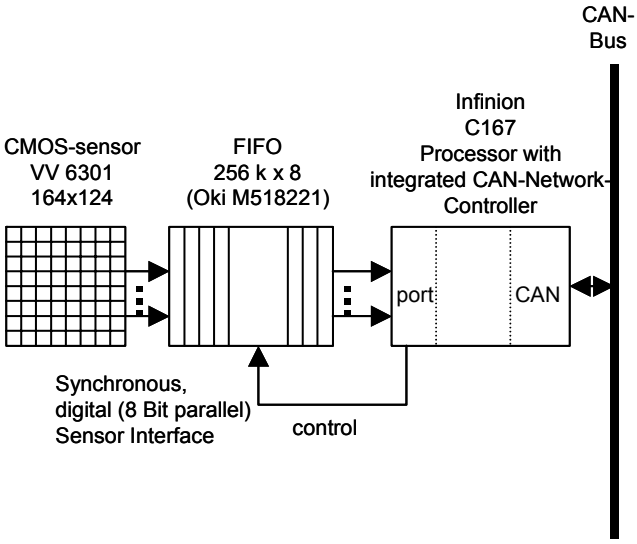


Fig. 6 The main components of ICU

The heart of the interface logic is a FIFO which decouples the fast synchronous operation of the sensor from the slower asynchronously operating processor interface. The only control line from the sensor to synchronize the processor with the sensor is a "start-of-frame" signal. The FIFO can buffer up to 13 frames which are delivered by the sensor with a maximum rate of 30 frames/sec. Currently, we only use 15 frames/sec, partly because of the port interface to the processor which is rather slow in copying data from the FIFO to the processor memory.

ICU has a CAN-Bus [14] network interface which is popular in automotive and industrial control systems and allows transfer rates up to 1 Mbit/sec with a protocol efficiency of about 50% payload. The maximal length of a single CAN-Bus message is 8 Bytes. The CAN-Bus was designed to reliably operate in environments exhibiting high electric noise, but obviously not for high speed data transfer. Therefore it would be impossible to transfer raw image data which would need around 2,3 Mbit/sec (160*120 pixels, 1 byte/pixel, 15 frames) even for the low resolution sensor. However, the pre-processed complete information to control the movement of the vehicle fits into a single CAN message and puts no significant load on the communication system.

At the moment ICU can be configured to either periodically publishing position and colour information on the CAN-Bus or to transmit a tracking position on demand. A small local software component, called event channel handler connects ICU to the publisher/subscriber middleware. The information delivered in a CAN message by ICU is depicted in Tab.1. ICU disseminates:

1. the position of the G-line within its window of perception,
2. the colour in separate values for red, green and blue and
3. brightness information

Byte	Content
1	x-position on upper line
2	x-position at center line
3	x-position at lower line
4	R-intensity
5	G-intensity

6	B-intensity
7	Brightness
8	Not used

Tab. 1 Payload of the CAN message

ICU is configured via the CAN-Bus. The configuration message defines the rate of the tracking information. If the configuration message is empty, i.e. it does not include any parameters, the camera returns a single message. In addition to the CAN interface, a serial interface is available for debugging purposes and for changing the flash memory of the micro-controller. Fig. 7 and Fig. 8 show the ICU prototype and the test environment depicting ICU hooked to a CAN analysis tool.

2.2.6 Conclusion and Future Work

Design of control systems for complex artifacts like autonomous vehicles, driver assisting cars, airplanes and industrial plants tend to become enormously complicated. One step towards solving the problem is to provide computational power at the sensors and actuators making them smart devices encapsulating all computations needed to convert a physical process value to an application related meaningful information. The autonomy and encapsulation properties encourage hardware/software co-design on the device level and component-based design methodologies [20] for putting the building blocks together. In the context of a project dealing with appropriate models and mechanisms to support decentralized systems composed from these components, we developed a couple of smart sensors.

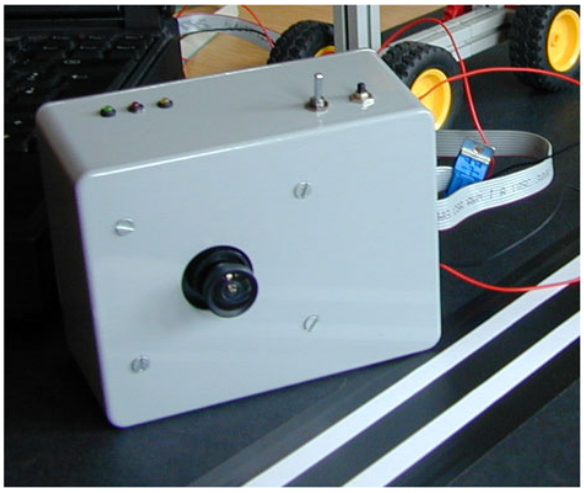


Fig. 7 ICU prototype

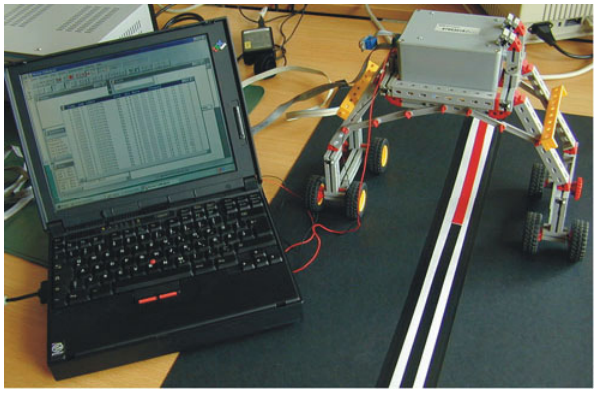


Fig. 8 Test environment for ICU

ICU is an optical sensor specialized on recognizing a tracking line for vehicle guidance. ICU provides tracking information via a CAN-Bus interface and the respective middleware, thus making it available to any entity which may exploit this information. This may be the smart motor controllers for directly steering the vehicle or a navigation system just logging the course. ICU is the first prototype of a low cost/low power optical sensor which is used in such an environment. Because ICU is adapted to a single application related function, resources like hardware, communication bandwidth and power consumption can be reduced to a minimum just as required by the application. ICU is designed for vehicle guidance based on tracking predefined lines and marks. At the moment, we use off-the-shelf processor boards [21] and only designed the glue hardware ourselves. In the future, we will be working in two directions: 1. we will design better and more robust algorithms to detect G-lines particularly detecting intersections and junctions of G-lines. 2. we will design a processor board which provides a faster interface to the optical sensor to enable more sophisticated vision tasks without substantially increasing power consumption which at the moment is around 250mA@5V for the entire ICU.

2.2.7 Acknowledgements

The work on ICU was partly sponsored by GMD AiS[3] and the University of Magdeburg. We want to thank Michael Mock (GMD) Edgar Nett (U. Magdeburg), Reiner Frings (GMD) and Martin Gergeleit (U. Magdeburg) for their co-operation and support. Particularly, we want to acknowledge Michael Wallner's effort to integrate ICU in the publisher/subscriber communication environment. This work was partly funded by the EU in the CORTEX[4] Project under the contract No.: IST-2000-26031.

2.2.8 References

[1] Alireza Moini: Vision Chips or Seeing Silicon, Third Revision, *http://www.eleceng.adelaide.edu.au/Groups/GAAS/Bugeye/visionchips/index.html*, March 1997

[2] R.A. Brooks: A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation, vol. RA-2, no.1*, March 1986

[3] J. Kaiser, M. Mock : Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN), *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99)*, Saint-Malo, France, May 1999

[4] J. Tanner, C. Mead: An integrated analog optical motion sensor, *in R.W. Brodersen & H.S. Moscovitz, editor, VLSI Signal Processing, II*, pp. 59{87. IEEE, New York, 1988.

[5] J. Stankovic: Misconceptions about Real-Time Computing, *IEEE Computer*, 1988

[6] H. Kopetz, E. Fuchs, D. Millinger, R. Nossal: An Interface as a Design Object, *Proc. Int'l Symp. on Object-oriented Real-Time Distributed Computing (ISORC-99)*, St. Malo, France 1999

[7] B. Oki, M. Pfluegl, A. Seigel, D. Skeen: "The information Bus®- An Architecture for Extensible Distributed Systems", *14th ACM Symposium on Operating System Principles, Asheville, NC*, Dec 1993,pp.58-68

[8] R. Rajkumar, M. Gagliardi, L Sha: " The Real-Time Publisher/Subscribe Inter-Process Communication Model for Distributed Real-Time Systems: Design and

[3] GMD AiS: German National Research Institute for Information Technology, Institute for Autonomous Intelligent Systems

[4] CORTEX: CO-operating Real-time senTient objects: Architecture and Experimental evaluation.

Implementation", *IEEE Real-time Technology and Applications Symposium,* June 1995

[9] K. Mori. Autonomous decentralized Systems: Concepts, Data Field Architectures, and Future Trends, *Int. Conference on Autonomous Decentralized Systems (ISADS93),* 1993

[10] N. Carriero, D. Gelernter: "Linda in Context", *Communications of the ACM, 32, 4*, April 1989

[11] Rene Meier: State of the Art Review of Distributed Event Models, *Tech. Report TCD-CS-2000-16*, Trinity College, Dublin Ireland, March 2000

[12] T. Harrison, D. Levine, and D. Schmidt: The design and performance of a real-time CORBA event service. In *Proceedings of the 1997 Conference on Object-Oriented* Programming Systems, Languages and Applications (OOPSLA), 1997

[13] Sun Microsystems. Java Distributed Event Specification. *Sun Microsystems, Inc*., July 1998, http://www.javasoft.com/products/javaspaces/specs

[14] Object Management Group. CORBAservices: Common Object Services Specification - chapter 4, Event Service Specification. *Object Management Group*, March 1995. http://www.omg.org/library/csindx.html.

[15] ROBERT BOSCH GmbH: "CAN Specification Version 2.0", Sep. 1991

[16] D. J. Musliner: CIRCA: The Cooperative Intelligent Real-Time Control Architecture, *PhD-Dissertation*, University of Michigan, 1993

[17] K. Z. Haigh: Situation-Dependent Learning for Interleaved Planning and Robot Execution, *Tech. Report CMU-CS-98-108,* CMU, February 1998

[18] M. Wallner: Ein Publisher/Subscriber-Protokoll für heterogene Kommunikationssysteme, *Master thesis*, University of Ulm, March 2001

[19] VLSI Vision Ltd: VV6300 Low Resolution Digital CMOS Image Sensor, Technical Description cd 34021-b.fm, UK, 1998, www.vvl.co.uk

[20] Siemens AG: 16 Bit Micro-controllers – C167 Derivatives, User Manual 03.96 Version 2.9, 1996

[21] C. E. Pereira , L. B. Becker, C. Villela, C. Mitidieri , J. Kaiser : On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects, *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC2001), Magdeburg, Germany, May 2001*

[22] Phytec: Mini-Modul 167 Hardware Manual, Phytec Messtechnik GmbH, Mainz, 1999, http://www.phytec.de

Chapter 3: Context Awareness

This chapter presents a position paper on "Context Awareness in CORTEX".

3.1 Introduction

CORTEX defines sentient objects, mobile intelligent software components that accept input from a variety of sensors allowing them to sense their environment before deciding how to react, by way of actuators. It is this awareness of, and interaction with the environment, which makes context awareness an important factor in sentient objects.

This paper examines the meaning of context and context awareness in terms of CORTEX and proposes a model for the development of context aware sentient objects. Our model draws on ideas from the domain of Context Based Reasoning.

3.2 Definition of Context

Before examining the role of context in CORTEX, a clear definition of what we understand by context is required. There are multiple definitions of context available in the literature. Schilit et al. [1] name the three important aspects of context as being where you are, whom you are with and what resources are nearby. They go further to divide context into the categories of *Computing context*, *User context*, and *Physical context*.

Dey et al. [2] provide a definition of context which aims to ease the enumeration of context for a given application scenario. Their definition of context follows "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." This definition of context is less specific than others, for example [1], in that it does not prescribe what aspects of context are important, rather leaving this to be done on an individual basis for each situation. For example user context may not be relevant to an autonomous sentient object, whereas the Quality of Service (QoS), or infrastructural context, provided by the underlying architecture is of more importance to the sentient object. In [2], 4 types of context are identified as being, in practice, more useful than others, these being *location, identity, activity* and *time*.

Chen et al. [3], define two aspects of context in mobile computing these being the characteristics of the surrounding environment that determine the behavior of an application and those that are relevant to the application, but not critical. These differences are taken into account in their definition of context, which follows "Context is the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user."

For the purposes of CORTEX we propose a definition of context as:

Any information sensed from the environment, which may be used to describe the current situation of a sentient object. This includes information about the underlying computational infrastructure available to the sentient object.

3.3 Definition of Context-Aware

A context-aware application is an application whose behavior is controlled by its context, to some degree.

Dey et al. [2] define a system as being context-aware if it "uses context to provide relevant information and/or services to the user, where relevancy depends on the users task". Their definition is influenced by the observation that a context aware system does not necessarily have to adapt to its context, it may just sense and display, or stores its context.

The definition of context provided by [2], is once again a user-centric view of context and context-awareness. In CORTEX, where we are dealing with autonomous, real-time sentient objects, the role of context in user interaction is not as important as say, interaction between sentient objects. Thus a possible definition for context-aware in CORTEX might be:

The use of context to provide information to a sentient object, which may be used in its interactions with other sentient objects or the environment, and/or the fulfillment of its goals.

3.3.1 Categories of Context-Aware Applications

Schilit et al. [1], in their work on context-aware applications, define four categories of context-aware application based upon two dimensions - whether the task is getting information or doing a command and whether it is affected manually or automatically. Their classification is illustrated in Table 1.

	Manual	**Automatic**
Information	Proximate selection and Contextual information	Automatic contextual reconfiguration
Command	Contextual commands	Context triggered actions

Table 1: Categories of context aware applications [1]

In terms of CORTEX, Schilit's classification is useful predominantly along the automatic axis (shaded in Table 1) due to the autonomous nature of sentient objects. Proximate selection is a user-interface technique and hence not useful in our existing definition of sentient objects. Likewise, Contextual Information and Commands deal with the parameterization of user commands effected manually and is not useful in our existing sentient object definition.

The other two categories of context-aware application defined by Schilit are as follows:

1. *Automatic Contextual Reconfiguration* refers to the addition of components, removal of components, or alteration of connections between components, depending upon context. In other words, the configuration of the system depends upon its context at a point in time.
2. *Context-triggered actions* are simple IF-THEN rules that dictate how a system adapts and reacts to a particular context.

The category of context-triggered actions is particularly useful in CORTEX, where we envision some form of inference engine based upon rules input from the Programming Model and operating in a similar manner to Context Based Reasoning. These concepts are explained and expanded upon in the remainder of this chapter.

3.3.2 Examples of Context-Aware Applications

A number of applications have been developed, primarily in research laboratories, which utilise contextual awareness to differing degrees in their operation. Some of these applications are briefly described here in order to demonstrate the state of the art in context-aware applications.

3.3.2.1 The Context Toolkit

The Context Toolkit developed by Dey et al. [4], is an architecture designed to assist in the development of context aware applications.

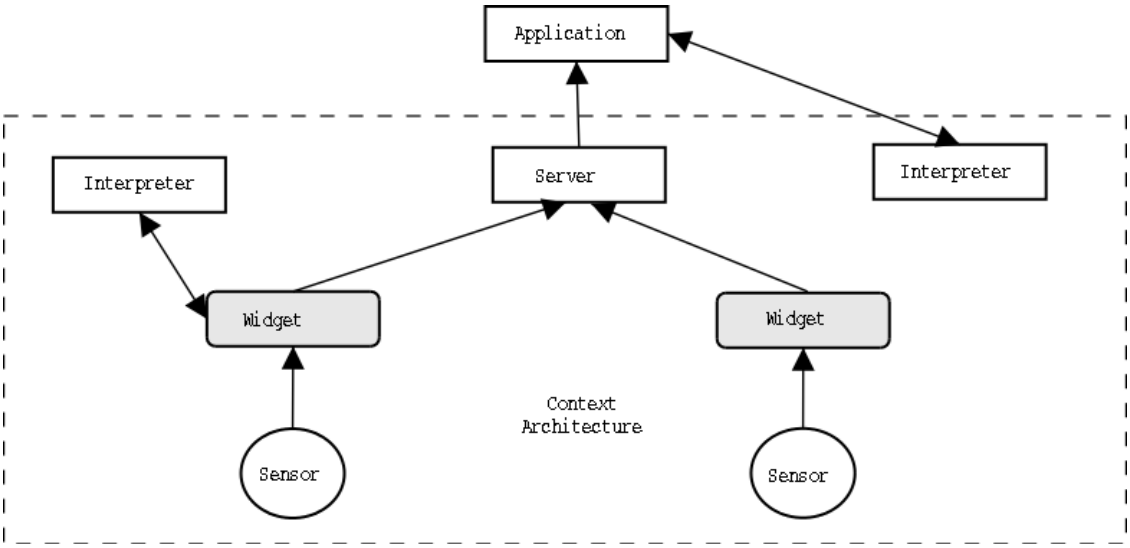


Figure 1: The Context Toolkit

The authors cite the problem that many context-aware applications in the past have been built in an ad-hoc manner and are heavily influenced by underlying technology. The major components of the Context Toolkit are illustrated in Figure 1, and their functions described below:

- A Context Widget hides the complexity of raw sensor data, abstracts information and provides re-usable components
- A Context Server serves to aggregate all the context information for a particular entity, from a variety of sensor sources.
- A Context Interpreter interprets context information and serves to separate the interpretation abstraction from the application, permitting reuse of interpreters.

The Context Toolkit facilitates the provision of contextual information to applications in an abstracted form. This allows application developers to use sensor data in a uniform way across applications and leverage off existing components in developing new applications.

The concepts of abstraction of sensor data, re-use of components and a separation of concerns between the sensing of context data and the use of context by a sentient object are all concepts that we expect to be addressed in developing context aware sentient objects.

3.3.2.2 TRIP

Significant in that the system uses a distributed component architecture, described as a Sentient Information Framework (SIF), the Target Recognition using Image Processing (TRIP) system uses vision based sensor technology to infer objects' approximate location, orientation and identity [5]. The SIF architecture upon which TRIP is based bears some analogies to CORTEX, in that it consists of a group of co-operating distributed software components. Context notifications are distributed asynchronously through the model by way of events. SIF is implemented in CORBA, and makes use of the CORBA Event Service, as well as the OMG Notification Service for event filtering.

In contrast to CORTEX, the TRIP system does not address timeliness or Quality of Service issues, and is not designed on the same scale as CORTEX.

3.3.2.3 Active Bat System

The active Bat system developed at AT&T Laboratories in Cambridge uses sensors to update a model of the real world, which is then used to write programs that react to changes in the environment [6].

The system is based on an ultrasound location sensing system, which uses wireless devices known as Bats to determine the 3-Dimensional position of objects. These Bats are carried by people within the experimental area, and are attached to static devices. The location sensing technology employed gives extremely fine-grained location data (accurate to less than one meter), which allows a detailed model of the world to be constructed. With such fine-grained location data, it is possible to determine if a person is standing or seated, and their orientation. Data accuracy is improved through filtering to minimize sensor errors.

The model of the world maintained through the Bat location data is used to provide a number of novel context-aware applications such as follow-me systems where a users computing desktop follows his movement and is displayed on the display closest to him.

The Active Bat system is notable in terms of CORTEX in that it uses a form of quality-of-service adaptation in the location update process. The base stations preferentially allocate location update resources to those Bats, which are changing their location (moving) frequently. This is achieved through a scheduling algorithm in the base station, which determines when a Bat will next be addressed to determine its location. Those Bats moving infrequently may go into a powered down mode to prolong the life of their battery.

3.3.2.4 Context-Based Reasoning (CxBR)

Context-Based Reasoning (CxBR) was introduced by Gonzalez [7], [8] as a concise but rich representation paradigm that could be used to model the intelligent behavior of opponents in simulations. The paradigm derives its name from the idea that the actions taken by an entity are highly dependent on the entity's current situation (context). Context-Based Reasoning is based on the following hypotheses:

1. Small, but important portions of all available environmental inputs are used to recognize and treat the key features of a situation
2. There are a limited number of things that can realistically take place in any situation
3. The presence of a new situation will generally require alteration of the present course of action to some degree

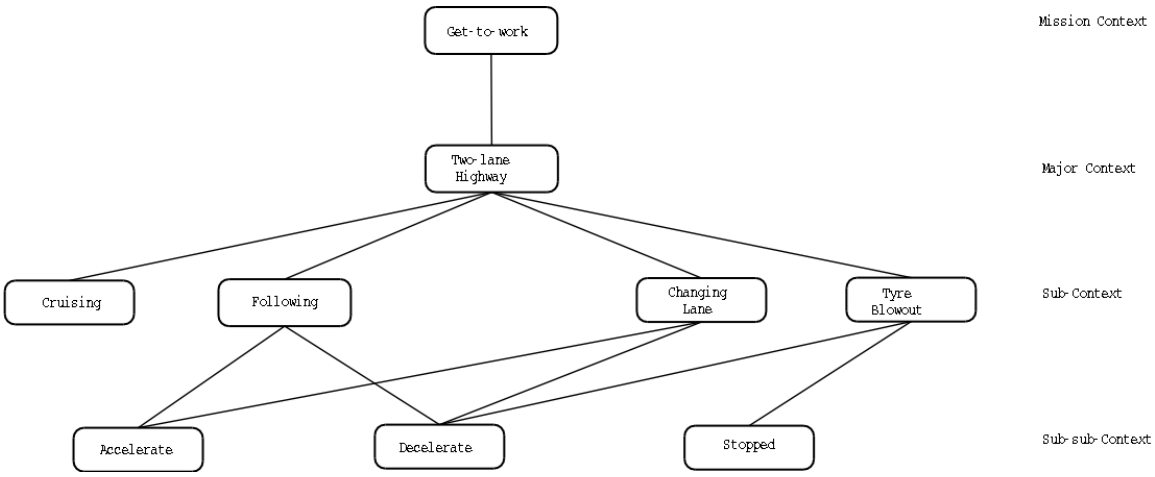


Figure 2: Situations for a sentient vehicle

Taking the example of the CORTEX Air Traffic Control application scenario, we can see that the above hypotheses hold true. Consider an aircraft sharing airspace with other aircraft. As the aircraft flies, it is accepting sensory input from a wide range of sensors throughout the craft. Fuel-levels, wind speed, altitude, tilt, yaw and proximity detection are a small subset of possible sensory input. Should another aircraft enter the airspace in a position, which violates a separation property, this will be determined by the proximity sensors. This small subset of

available environmental inputs has been used to recognize the fact that another aircraft has violated the separation property of our craft, as in hypothesis (1).

Continuing with our example of the aircraft, and hypothesis (2), it is unlikely that the situation whereby the separation property of the aircraft is violated can occur whilst the aircraft is stationary on the apron refueling. A more likely prospect in this situation is the embarkation of passengers.

The presence of the new situation, violation of the separation property of our aircraft, will require some alteration of the present course of action as hypothesized in (3). In this case, it may include altering the present action (cruising) to climbing, diving or turning.

Following the hypotheses above, it can be seen that by associating specific situations and the actions to take in these situations, with contexts, the identification of a situation is simplified (since only a subset of situations are possible within a given situation). The association of courses of action with specific situations leads to the influence of behavior by context.

3.3.2.4.1 Example of Context Based Reasoning

We may apply Context-Based Reasoning to the CORTEX application scenario of co-operating cars where the car control system is always aware of its geographical and traffic context [9]. In this application scenario, we can define a number of situations in which the car may find itself in during the accomplishment of its goals. A possible subset of these situations is illustrated in Figure 2.

In this example, there are 4 levels of situation or 'contexts' in which the car may be at a point in time. The Mission Context is the overall objective of the scenario. In our example, the mission of the car is to get to work. This context will be active throughout the scenario.

A Major Context is an operation that needs to be undertaken in order to complete the mission. In our example, the car needs to travel down a two-lane highway in order to accomplish the mission of getting to work. Another Major Context in our example may be negotiating a 4-way stop. Only one Major Context may be active at any point in time.

A Sub-Context is a certain behavior that is associated with a Major Context. Each Sub-Context may be associated with more than one Major Context. So the car in our example may be in the cruising Sub-Context while on a two-lane highway or on a single-lane road.

Finally, a Sub-sub-context is a low level action, which is carried out as part of a Sub-Context. In our example, passing consists of accelerating to pass a slower car, then decelerating back to the speed limit, once the slower car has been passed.

3.4 Issues in Context-Aware Computing

There are a number of issues that need to be addressed in context aware computing including inaccuracies in sensed data, how context information is modeled and privacy and security concerns.

3.4.1 Sensor Failure Modes

Sensor data always has a degree of uncertainty associated with it, due to noise or sensor inaccuracies, and any system utilizing such data, must take into account the precision of the sensor data. Data fusion, the combination of data from a variety of sensor sources can be used to reduce the uncertainty of a single data source and is often used in navigation systems.

Another technique used to reduce the inaccuracies inherent in sensor data, is that of digital filtering techniques such as Kalman filtering. Used in combination with data fusion, such techniques may be used to provide sensor data with a certain level of certainty.

3.4.2 Representation and Modeling of Contextual Information

There is no existing standard for the representation of contextual information, but XML provides an expressive and highly flexible means to model interrelated data. It also enables access and reasoning about the data stored. XML seems to be a promising technology for context representation and modeling.

```
<vehicle>
   <required>
      <direction> 227
         <speed> 50
            <time> 09:58

<body>
```

The XML snippet above demonstrates how the context of a vehicle may be stored. In the example, the vehicle is traveling due south-west (227 degrees), at 50 miles per hour at 09:58.

A representation scheme based upon XML has the advantages of being eminently extensible, and able to deal with heterogeneous context data.

3.4.3 Privacy and Security

Since the context of an entity includes both the identity and the location of the entity, contextual information has the potential to be highly sensitive, especially if the information pertains to a person. As a result, privacy and security requirements of contextual information need to be addressed. Security requirements can clearly be seen in the Air Traffic Control application scenario of CORTEX where the location and identity of aircraft is likely to be sensitive information.

Standard data privacy and security measures will need to be incorporated into CORTEX, with no exceptional requirements identified at this point.

3.4.4 Problems Inherent in the Use of Distributed Context-Awareness for Interaction

In their essay on distributed context-aware systems, Benerecetti et al. [10] argue that in a distributed system, an entity must not only be aware of its own context, but must take into account the fact that other entities operate in different contexts, and this has important consequences in the interaction of such entities. They argue that an objective notion of context relies on the assumptions that

1. The entity exists in an objectively definable environment
2. There is a single list of relevant features that an entity needs to sense from the environment
3. Each feature may be assigned an unambiguous value

In a distributed system such as that envisaged under CORTEX, these assumptions are not necessarily true. The first assumption does not account for the fact that different sentient objects may represent different features of the same physical environment. Even if the sentient objects share the same physical environment, if their representation of that environment differs, they may not be considered to share the same environment.

The second assumption does not hold since there is no list of contextual features that are relevant for every sentient object in a sentient system.

The third assumption fails because of the fact that different sentient objects may interpret the same environmental feature in different ways, or at different levels of granularity (for example one sentient object may interpret its location as 'Trinity College Dublin', whilst another may interpret it as 'O'Reilly Institute, Trinity College Dublin').

When dealing with context awareness in a distributed system such as CORTEX, it is essential that we bear in mind that in addition to a sentient object being aware of its own context, it needs to be aware of the fact that other sentient objects exist in other contexts (or have different representations of the same context). This fact will have implications on interactions between sentient objects, especially those mediated by contextual awareness.

3.5 Context-Awareness in CORTEX

Following the definition of a sentient object in CORTEX as proposed by [11], and examining how contextual information might be used to control a sentient object, we propose the model illustrated in Figure 3. The model consists of three components, or functions within the sentient object.

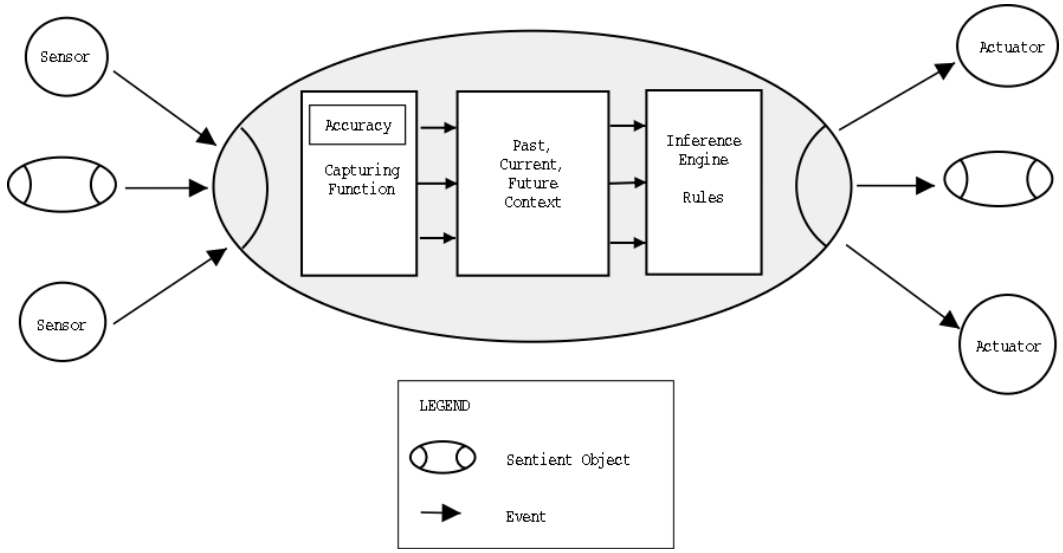


Figure 3: Sentient object internals

3.5.1 Context Awareness in Sentient Objects

In order for context awareness to be used in CORTEX, the use of context awareness has to be specified at the level of individual sentient objects. The three components to facilitate the use of contextual information in a sentient object are a capturing function, a representation function and an inference function.

3.5.1.1 Capturing Function

The capturing function deals with event data entering the sentient object from sensors and from other sentient objects and performs a function similar to that of a Context Widget by masking the complexities of raw sensory information. All sensory information has accuracy and probability associated with it and this is taken into account right at the beginning, when the sensory data is captured. It is at this point where techniques such as Kalman filtering may be applied to reduce errors inherent in the sensor data, and to assign a level of confidence to the accuracy of the data.

3.5.1.2 Current and Past Context Representation

The task of representing contextual information is dealt with in this component. Each sentient object will have different requirements as to what contextual information is important to it,

and how this information is represented. Contextual history is stored, since often behavior may be inferred, in the absence of current context information, based upon past context information. Contextual future is also extrapolated from the present and is stored to assist in reinforcement of learning behavior in the inference engine. The inference engine may also create or adapt rules based upon past context information.

3.5.1.3 Inference Engine

The Inference Engine component will likely contain a form of rules-based engine, which dictates how the sentient object acts in a specific context. It is expected that this will be based upon context-based reasoning, where predefined situations are associated with specific actions.

This is similar to the approach of [12], where actions are assigned to perceived contexts, and may be specified on the actions of entering, leaving and remaining in particular contexts.

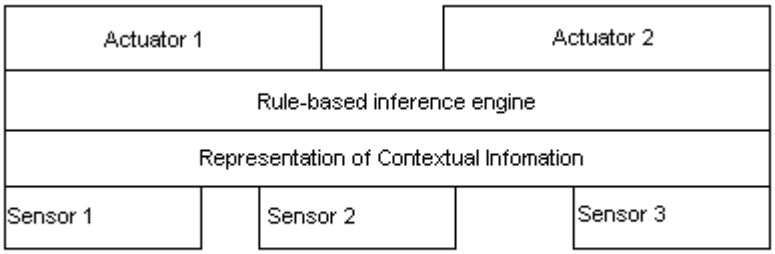


Figure 4: A layered architecture

Figure 4 illustrates the layered architecture of the proposed approach to context aware sentient objects. It is important to note that the sensors and actuators may accept and produce both real world events and software events.

3.5.2 Infrastructural Awareness as a Factor in Context-Awareness

An important facet of contextual awareness in a CORTEX scenario is awareness of the infrastructure. Infrastructural awareness in CORTEX is primarily concerned with the quality-of-service as provided by the infrastructure. In the WAN of CAN's architecture proposed by CORTEX, the heterogeneity of the underlying networks coupled with the mobility of sentient objects means that the infrastructure available to a sentient object will be dynamic. CORTEX recognizes this and strives for a highly adaptive behavior accounting for dynamically varying network parameters. Adaptive behavior in this sense includes an adaptable model of synchrony, where levels of synchrony may be specified on a per group basis. Such adaptive behavior in CORTEX makes awareness of the underlying quality-of-service essential.

It is this infrastructural awareness as a major factor in context-awareness in CORTEX that sets CORTEX apart from existing context-aware applications.

3.5.3 Context-Awareness in the Interaction Model

In CORTEX, interaction between large numbers of sentient objects is proposed. Centralized co-ordination amongst sentient objects in networks of such scale is not a viable option due to considerations such as the presence of a single point of failure and the inability of the centralized model to scale well. A model of co-ordination based upon local knowledge and decision-making is more appropriate to such large networks of sentient objects.

Following a localized co-ordination model for sentient objects in CORTEX, it is proposed that the further apart entities are from each other, the less interest they have in interacting with each other. Thus interaction becomes a function of proximity.

This concept of localized interaction forms the basis of a type of interaction and coordination known as *stigmergy* where interaction between entities does not require explicit communication between them, and is mediated by the environment. Stigmergy allows insect colonies to achieve highly coordinated behavior through communication via the environment and has successfully been applied to a number of robotic and routing applications. Stigmergic coordination, since the environment mediates it, has awareness of the environment or context awareness, at its core. Stigmergic coordination is one possible mechanism for the coordination of sentient objects, which is currently under investigation [13].

3.5.4 Relation to the Programming Model

The integration of the use of contextual awareness into the CORTEX Programming Model poses some real challenges. Since the Programming Model is not yet fully defined, the ideas offered here on integrating the use of contextual awareness are bound to be refined as the Programming Model matures.

Initial thoughts on how contextual-awareness may be incorporated into the Programming Model are cantered on three major steps in the programming of sentient objects. These steps are outlined below.

1. **Specification of what contextual information is of importance to the sentient object**

Each sentient object will have different requirements when it comes to what aspects of its context are important. For a sentient vehicle, location is likely to be of prime importance, as is the relative position of other vehicles in close proximity. This step may involve the development of an XML Document Type Definition (DTD) to describe the entity's representation of its context. Alternatively, CORTEX could provide skeleton DTD's for a subset of sentient objects, with the programmer being able to customize and extend these.

2. **Specification of important situations (contexts), which the sentient object, may exist in**

This step is a form of expert knowledge capture in that it deals with the identification of states in which the sentient object may find itself, and specifies how these states are recognized, based upon contextual information. Staying with the sentient vehicle example, important situations a vehicle may find itself in, such as cruising, changing lane, overtaking and how to recognize these situations, is captured at this point.

3. **Specification of the sequence of actions to be performed in each situation**

By specifying the actions to be undertaken in each situation, the behavior of the sentient object may be controlled by its context. This idea is similar to the scripting primitives described in [12] where actions are associated with contexts and are further refined to be associated with entering a certain context (situation), leaving a context and being in a context.

3.5.5 Relation to the Event Service

CORTEX defines an anonymous, generative communication paradigm in the form of an event based communication model. An event based communication system will facilitate the asynchronous notification of abstracted sensor information to interested sentient objects in a sentient system. Importantly, the use of event-based communication has the potential to handle different interaction patterns amongst sentient objects and will allow the evolution of sentient objects to accommodate new and diverse sources of contextual information.

The integration of context awareness into sentient objects in CORTEX is not expected to have any exceptional additional requirements from the event service. Specific context events will

be developed, but this will probably be a part of the overall development process of a sentient object.

3.5.6 Evaluation Criteria

We intend to evaluate the ideas offered within this paper under a set of criteria in order to assess their suitability for CORTEX.

3.5.6.1 Context Based Reasoning

The concepts behind Context Based Reasoning will be applied to CORTEX application scenarios in order to evaluate whether the paradigm is suited to sentient objects. The evaluation will take the form of identifying important situations for sentient objects within each application scenario, and modeling these as a hierarchy of contexts as in Context Based Reasoning. Actions to be undertaken in each context and rules for transition between contexts will also be developed and the approach will be evaluated on the basis of the **efficiency**, **ease of development**, and **completeness** of the resulting model. A major goal is to assess how accurately Context Based Reasoning can model the behavior of a sentient object.

3.5.6.2 Proposed Architecture

The proposed architecture and consisting of a capturing function, representation function and inference function will be evaluated to see whether the approach is viable in terms of the use of context awareness to direct the behavior of sentient objects.

Importantly, the suitability of XML as a means to represent contextual information will be evaluated through the development of DTD's for sentient objects identified in CORTEX application scenarios. The ability to represent contextual information with completeness and the ability to reason about this information will be evaluated through examples drawn from application scenarios.

The inference function is based on Context Based Reasoning and will consist of IF-THEN rules. The ability of this rule based approach to direct the behavior of sentient objects based upon their context will once again be evaluated on the basis of examples from CORTEX application scenarios.

3.6 Conclusions

In this paper we have introduced context awareness as an important factor in the development of sentient objects, which interact with their environment, and are controlled in part by environmental factors. We defined the terms *context* and *context aware* in terms of sentient objects in CORTEX, and went on to propose a high level model for context aware sentient objects. Challenges posed by distributed context awareness were also examined. Initial ideas on the relation between context awareness and the CORTEX programming model were then offered.

It is our position that context awareness plays an important role in the interaction of sentient objects through the environment as described in CORTEX. Awareness of the environment is a prerequisite for any such interaction.

3.7 Future Work

The high level model of context aware sentient objects needs to be refined and it is expected this will be achieved through the application of ideas proposed in this paper, to CORTEX application scenarios. It is hoped that by examining context awareness in each of the proposed application scenarios, a generic specification of context awareness in CORTEX will be achieved.

3.8 References

[1] Schilit, B., Adams, N. Want, R. *Context-Aware Computing Applications* 1st International Workshop on Mobile Computing Systems and Applications. 1994 pages 85-90

[2] Anind K. Dey and Gregory D. Abowd *Towards a Better Understanding of context and context-awareness.* Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, June 1999.

[3] Guanling Chen and David Kotz *A Survey of Context-Aware Mobile Computing Research.* Department of Computer Science, Dartmouth College Technical Report TR2000-381.

[4] Anind K. Dey, Daniel Salber, Masayasu Futakawa, and Gregory D. Abowd *An architecture to support context-aware applications* Technical Report GIT-GVU-99-23, Georgia Institute of Technology, College of Computing, June 1999.

[5] Diego Lopez de Ipina *Building Components for a Distributed Sentient Framework with Python and CORBA* In Proceedings of the 8th International Python Conference*, Arlington, VA, USA. 24-27 January, 2000*

[6] Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, Andy Hopper *Implementing a Sentient Computing System* IEEE Computer, August 2001

[7] Avelino J. Gonzalez *Context-based Representation of Intelligent Behaviour in Simulated Opponents* University of Central Florida, http://isl.engr.ucf.edu/publications/CxBR.html

[8] Fernando G. Gonzalez, Grejs Patric, Avelino J. Gonzalez *Autonomous Automobile Behaviour through Context-based Reasoning*, University of Central Florida, http://isl.engr.ucf.edu/publications/CxBR.html

[9] CORTEX project partners *CORTEX Definition of Application Scenarios*, v1.0, October 2001.

[10] Massimo Benerecetti, Paolo Bouquet, Matteo Bonifacio *Distributed Context-Aware Systems* Special Issue of Human-Computer Interaction, Volume 16 2001

[11] Adrian Fitzpatrick *CORTEX Sentient Object Specification* Trinity College Dublin January 2002

[12] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, Walter Van de Velde *Advanced Interaction in Context* International Workshop on Handheld and Ubiquitous Computing 1999 pages 89-101

[13] Greg Biegel *Stigmergy in CORTEX* Trinity College Dublin, February 2002.

Chapter 4: Event-Based Communication Model

This chapter presents two technical reports addressing the requirements of event-based programming models. The first report describes the programming model of event-based middleware designed to provide inter-object communication pattern for mobile objects in a wireless environment utilizing an ad-hoc network model. The second report presents a survey of existing event systems structured as a taxonomy of distributed event-based programming systems. Both reports have been accepted for publication at the International Workshop on Distributed Event-Based Systems [7], which is to take place in July 2002.

4.1 STEAM: Event-Based Middleware for Wireless Ad Hoc Networks

René Meier and Vinny Cahill

Department of Computer Science, Trinity College Dublin, Ireland

{rene.meier, vinny.cahill}@cs.tcd.ie

Abstract - With the widespread deployment and use of wireless data communications in the mobile computing domain the need for middleware that interconnects the components that comprise a mobile application in distributed and potentially heterogeneous environments arises. Middleware utilizing an event-based communication model is well suited to address the requirements of the mobile computing domain, as it requires a less tightly coupled communication relationship between application components compared to the traditional client/server communication model. This is particularly useful with the use of wireless technology, where communication relationships amongst application components are established very dynamically during the lifetime of the components. Recent research in the area of event-based middleware for the mobile computing domain focuses on infrastructure network models for wireless data communication. In this paper, we present STEAM, an event-based middleware service that has been specifically designed for wireless local area networks utilizing the ad hoc network model. We argue that an implicit event model is best suited for the envisaged ad hoc environment and present our approach of exploiting a novel combination of three different types of event filter to address the problems related to the dynamic reconfiguration of the network topology as well as their impact on the scalability of a system and the timely delivery of events.[5]

4.1.1 Introduction

The widespread deployment and use of wireless data communications in the mobile computing domain is generally recognized as being the next major advance in the information technology industry. Both mobility and wireless networking represent key enabling technologies underlying the vision of *ubiquitous computing* [1], where interconnected computers will be embedded in a wide range of appliances ranging in size from door locks to vehicle controllers performing tasks, such as automatically opening doors and routing vehicles to their intended destinations, on behalf of their human users. The event-based communication model represents an emerging paradigm for middleware that asynchronously [2] interconnects the components that comprise an application in a potentially distributed and heterogeneous environment, and has recently become widely used in application areas such as large-scale internet services and mobile programming environments [3]. Event-based communication is well suited to address the requirements of the mobile computing domain [4]. It avoids centralized control and requires a less tightly coupled communication relationship between application components compared to the traditional *client/server communication model*. This is particularly useful with the use of wireless technology in the mobile computing domain, where communication relationships amongst heterogeneous application components are established very dynamically during the lifetime of the components.

[5] The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET programme of the Commission of the EU under research contract IST-2000-26031 (CORTEX).

Mobile computing environments can utilize either the *infrastructure* or the *ad hoc network model* for wireless data communication [5]. The infrastructure network model exploits *access points* to establish communication relationships among mobile application components and to coordinate their transmissions. An access point is analogous to a base station in a cellular communications network forming groups of mobile application components that are under its direct control. Access points may be connected to a fixed network, such as a company intranet or the Internet, and act as a portal allowing the components under its control to connect to the fixed network. In contrast, the ad hoc network model allows application components to communicate with each other without the aid of access points or a fixed network. Any application component can establish a direct communication relationship with any other application component without having to channel the transmission through an access point (provided there is no network partition). This allows application components to communicate and collaborate in a spontaneous manner in the absence of a conventional fixed network.

Several middleware services utilizing the event-based communication model have been developed thus far by both industry [6] and academia [2], [7]. Most of these assume that the application components comprising an application are stationary and that a fixed network infrastructure is available to facilitate communication. They do not address the problems introduced by mobile application components and wireless network models, related to the dynamic reconfiguration of the network topology. Recently, some research has been done to support mobile computing in event-based middleware [4], [8]. However, this work has focussed on the infrastructure network model for wireless data communication assuming middleware components acting as event dispatchers being available. In this paper, we present STEAM (Scalable Timed Events And Mobility), an event-based middleware service that has been designed for the mobile computing domain. Specifically, it is intended for IEEE 802.11-based wireless local area networks (WLAN) utilizing the ad hoc network model. The next section introduces event-based middleware and provides a preliminary description of the differences between the traditional distributed computing domain that relies on fixed network infrastructure and the mobile computing domain. Section 4.1.3 outlines the design restrictions that are imposed on STEAM by the ad hoc network model. In section 4.1.4, we present our approach to overcome these restrictions and discuss how STEAM exploits a novel combination of three different types of event filter to address issues that are specific to the mobile computing domain and the ad hoc network model and their impact on the scalability of a system and the timely delivery of events. Finally, section 4.1.5 concludes this paper by summarizing our work and outlining the issues that remain open for future work.

4.1.2 Event-Based Middleware

An event system is an application that uses event-based middleware to allow the components that comprise the application to interact through *event notifications*. Event notifications, or simply events, contain data that represent a change to the state of the sending application component. They are propagated from the sending application components, called the producers, to the receiving application components, called the consumers. Events typically have a name and may have a set of typed parameters whose specific values describe the specific change to the producer's state. In order to receive events, event consumers have to *subscribe* to the instances of events in which they are interested; they are said to register interest in events. Once consumers have subscribed to events, they receive all subsequent events until they *unsubscribe* (de-register). An event system using event-based middleware, which is also called an *event service*, may consists of a potentially large number of application components, or entities, that produce and consume events. In conventional distributed event systems, entities are located on a number of physical machines that are interconnected by means of a fixed network infrastructure through which event-based communication takes place. In contrast, the STEAM event service is intended for the mobile computing domain where entities reside on mobile computing devices utilizing wireless networks to interact.

4.1.3 STEAM Design Issues

When designing event-based middleware or indeed middleware in general, the specific restrictions imposed by the environment for which the middleware is intended needs to be considered. This is particularly important when designing an event service for ad hoc networks, since the environment places especially strict limitations on the middleware due to the lack of any network infrastructure. Traditionally, event services exploit a range of services and protocols including lookup and naming services that are utilized by consuming entities to locate producing entities in order to route event subscriptions to them. Some event services, such as the CORBA notification service [6], use the service of intermediate middleware components for producing and consuming entities to communicate with each other. These services and intermediates must be accessible by the entities in the event system and are often implemented as independently running middleware components located in separate address spaces potentially on remote physical machines. Most significantly, utilizing such middleware components is not practical in an environment that is based on the ad hoc network model for a number of reasons. Middleware components cannot be located on separate physical machines as this approach assumes infrastructure that is inherently omitted in ad hoc environments. It can be argued that such middleware components may be co-located with the mobile entitles sharing the same physical machine. However, this does not overcome the most significant restriction of ad hoc environments, the limitation of the area that can be covered by mobile application components using wireless transmitters. Using a middleware component acting as intermediate requires all entities in a system being able to communicate with it at any given time. This is unlikely in an ad hoc environment as entitles may be distributed over a potentially large geographical area and thus are unlikely to be able to maintain a permanent communication link to the intermediate. In conclusion, the main restriction on the design of STEAM imposed by the ad hoc environment is the omission of application components that provide system-wide services.

4.1.4 The STEAM Event Service

In this section, we present our approach to overcome the restrictions imposed on event-based middleware by the mobile computing domain due to mobile application components, the wireless ad hoc network model, and the resulting limitations identified in the previous section. STEAM exploits a novel approach of combining three different types of event filter to address the dynamic aspect of the network topology. Our approach to event filtering supports the predictable behaviour of the event service, which is essential for the scalability of a system and the timely delivery of events.

STEAM has been designed for mobile environments, specifically with the traffic management application domain in mind. In this domain, we envisage application scenarios that include a large number of entities representing real world objects using wireless technology and the ad hoc network model. Many of these entities may represent mobile objects including cars and ambulances; other entities may represent object with a fixed location, such as traffic signals and lights. All entities interact using event-based communication in order to exchange information on the current traffic situation. For example, a traffic signal may propagate a change to the speed limit due to road conditions to approaching cars. Another example may involve an ambulance disseminating its location to the cars in its vicinity for them to yield the right of way. Our work is motivated by the hypothesis that in this type of application scenario entities are most likely to interact once they are in close proximity. Reflected on event-based middleware, this means that the closer consumers are located to a producer the more likely they are to be interested in the events propagated by that producer. Significantly, this implies that events are valid within a certain geographical area surrounding a producer. An example scenario demonstrating such behaviour would be a traffic light disseminating its status and cars being interested in receiving these events only if they are located within a certain range of the light. Our approach to propagating events within a certain area surrounding producers limits forwarding of event messages, and therefore reduces the usage of communication and

computation resources, which are typically scarce in mobile environments. Furthermore, we expect the limitation of event forwarding to reduce the susceptibility of an event system to radio frequency interference. Subsequently, the behaviour of communication connections between entities becomes more predictable, allowing the system to support reliable event delivery.

4.1.4.1 The Event Model

We have identified three distinct event models that may be implemented by an event service, namely peer to peer, mediator, and implicit [9]. Both peer to peer and mediator-based event models rely on system-wide services for entities to locate intermediate components or indeed each other. As argued in section 4.1.3, such approaches are not suitable for environments based on the ad hoc network model. Thus, the STEAM event service implements an implicit event model that allows consuming entities to subscribe to particular event types rather than at another entity or a mediator, without having to rely on system-wide services to locate entities or mediators, or on intermediate middleware components through which entities interact.

4.1.4.2 Proximity Group Communication

Group communication [10] has been recognized as a natural means to support event-based communication models [11]. Communication groups provide a one to many communication pattern that can be used by producers to propagate events to a group of subscribed consumers. Although there are other approaches to support message-oriented communication models, including distributed transactions, remote method invocation, and higher-level approaches such as workflow systems, group communication has been identified as the most suitable approach [11]. STEAM exploits a proximity-based group communication service [12] as the underlying means for entities to interact. Proximity groups have been identified as a useful communication paradigm for mobile applications utilizing wireless networks [12], based on the location of the application components. It allows mobile application components to discover each other once they are within the same geographical area using beacons. Significantly, this notion of proximity groups involves both geographical and functional aspects, i.e., the geographical area and the name of the group, to apply for group membership. An application component must firstly be located in the geographical area corresponding to the group and secondly be interested in the group in order to join. In contrast, classical group communication defines groups solely by their functional aspect. STEAM defines both the functional and the geographical aspect that specifies a proximity group. The functional aspect represents the common interest of producers and consumers based on the type of information that is propagated among them. The geographical aspect outlines the scope within which the information is valid, i.e., the limits within which it is propagated. Furthermore, STEAM exploits the message delivery semantics associated with proximity groups to provide end-to-end guarantees when delivering events.

4.1.4.3 Event Filtering

An event system may consist of a potentially large number of producers, all of which produce events that may contain different information. Therefore, the number of events propagated in an event-based system may be quite large. However, a particular consumer may only be interested in a subset of the events propagated in the system. *Event filters* provide a means to control the propagation of events. Ideally, filters enable a particular consumer to subscribe to the exact set of events that it is interested in receiving. Before events are propagated, they are matched against the filters and are only delivered to consumers that are interested in them, i.e., for which the matching produced a positive result. STEAM supports three different types of event filter, which are subject, proximity and content filter. The combination of these three types of event filter is specific to the mobile computing domain for which STEAM is designed.

Events propagated by STEAM consist of a name and a set of typed parameters. The name represents the type of an event classifying its structure and the parameters contain the values that describe a particular instance of an event. A common vocabulary is used by producers

and consumers in order to agree on the name of a specific event type. A subject filter describes the particular type of the events in which a consumer is interested and hence is matched against event types. It is mapped onto a proximity group and corresponds to the proximity group's functional aspect. A proximity filter corresponds to the geographical aspect of a proximity group describing the geographical area within which events of a specific type are valid. Thus, a proximity filter specifies the scope within which events are disseminated. In principal, either a consuming or a producing entity may specify a proximity filter. However, we believe that in many application scenarios it is the producer that would specify the geographical area within which the generated events are valid. For example, a traffic light propagating events containing the status of its light to approaching cars defines its proximity filter based on the location of the next traffic light and on the local speed limit. STEAM allows producing entities to define their proximity filters, which are deployed when event types are announced. In summary, subject and proximity filters allow a consumer to express interest in a set of events based on their type and the geographical area in which they are propagated. Content filters contain a filter expression that can be matched against the values of the parameters of an event. They are specified using filter expressions that may contain equality, magnitude and range operators as well as ordering relations to describe the constraints of a consumer, similar to the semantics of content filter expressions described in [13]. Subject and proximity filters are applied on the producer side. Events are only propagated to a consumer if both filters match. In contrast, content filters are located at the consumer side and will be evaluated when an instance of an event is received to determine whether or not to deliver it to the application. This approach does not require consumers to pass content filters to producers when subscribing. Most significantly, they need not be forwarded to producers when a consumer changes its location from one geographical area to another. This results in a simple approach to maintaining the subscriptions and content filters in a system that accounts for the dynamic reconfiguration requirement of the system that arises due to the mobility of the entities.

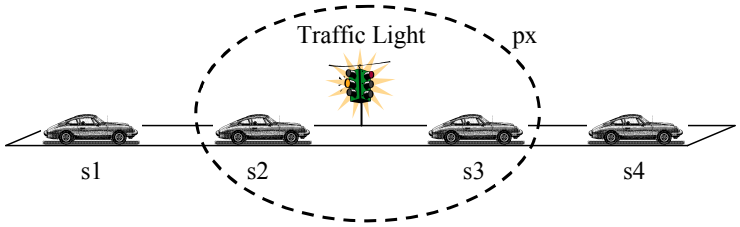


Figure 1.Traffic Light Application Scenario.

The traffic management application domain where (potentially driverless) vehicles are moving through a city interacting with each other and the environment offers a range of example application scenarios demonstrating the exploitation of subject, proximity, and content filters. Figure 1 depicts a traffic light application scenario in which a car is driving along a road acting as a consumer of events, thereby passing through the proximity *px* of a traffic light acting as event producer. The generated events are of type "traffic light" and their parameters contain the status of the light and its location. The car is shown at stages *s1* to *s4* of its journey and is assumed to have an associated subject, proximity, and content filter. The subject filter expresses interest in events of type "traffic light", the proximity filter defines *px*, and the content filter compares the car's previous and current position with the location of the traffic light stating that "events of the traffic light towards which I am driving" should be delivered. The results of matching events against each of these filters at the different stages of the car's journey and their effect on propagation and delivery of events are summarized in Figure 2. At stage *s1* of its journey, the car subscribes to the traffic light events deploying the corresponding filters. All subsequent events produced by the traffic light will be matched against the car's filters and will be delivered to it if all three filters produce a positive match. Figure 2 shows that at stage *s1* (after the car has subscribed) traffic light events are neither propagated nor delivered to the car as it is located outside the traffic light's proximity causing the proximity filter to produce a negative match. Traffic light events are propagated to the car

at journey stages *s2* and *s3* where both the subject and the proximity filter match as the car has subscribed and is located within the traffic light's proximity. However, they are only delivered to the car's application at *s2* as this is the only stage where the content filter matches as well. The content filter does not match at stage *s3* since the car has passed the light's location. No events are propagated at stage *s4* because the car has left the proximity of the traffic light. When continuing its journey, the car is likely to enter the proximity of another traffic light, whose events it will receive if the content filter matches without explicitly issuing another subscription.

Stage	Filter			Events	
	Subject	Proximity	Content	Propagated	Delivered
s1	match	no match		no	no
s2	match	match	match	yes	yes
s3	match	match	no match	yes	no
s4	match	no match		no	no

Figure 2. Matching of Events Against Filters.

Scalability has been identified as one of the key distributed computing problems that remains open in message-oriented middleware [11]. Applications exploiting event-based middleware may be large, they may consist of a very large number of entities and subscriptions. The ability of such as system to easily cope with the addition of entities and subscriptions is essential. This is particularly crucial in event systems that are based on the ad hoc network model as their scale typically changes dynamically over time. For example, a traffic management application dealing with the entities representing the cars driving through a city must be able to handle the cars involved regardless of their number, which may range from relatively small during the night to very large during rush hours. STEAM addresses scalability in two ways. It enhances the ability of a system to grow by limiting the propagation of events in space hence reducing the need for events to be forwarded and by utilizing a combination of filters that allow events being matched efficiently. STEAM allows an application to outline the scope in space (the proximity) within which some information (the event type) is valid. As events only need to be forwarded within the limits of their respective proximity, adding an entity affects only the area in which the entity is located. A joining entity causes other entities in the same proximity to reconfigure, i.e., routing and subscription information need to be updated. However, it does not require entities outside the proximity to adjust. Using a combination of subject and content filters within the scope defined by a proximity filter enables a system to combine the advantages of both. Subject filters support the deployment of an optimal matching algorithm. A simple table lookup on the producer side based on the subject results in a constant time algorithm. Content filters support the deployment of filter expressions that can be matched against different parameters of an event. Applying content filters on the consumer side instead of the common approach of deploying them on the producer side results in the distribution of the matching load from a single producer to a number of consumers. Therefore, each consumer has to deal with a small number of content filters, e.g., potentially one per subject, compared to a producer having to match the potentially arbitrary large number of content filters of every subscribed consumer. This approach causes additional overhead due to the propagation of unwanted events to consumers, which are then discarded by the content filter. However, the additional overhead is minimal as the underlying proximity group communication mechanism uses an approach that is based on flooding the area of the proximity with messages at the network layer in order to provide reliable message delivery semantics.

STEAM allows applications to define delivery deadlines and to assign them to specific events. A dispatcher then exploits these deadlines to determine the time to deliver the corresponding events to the subscribed consumers. For the dispatcher to enforce the timely delivery of events the event service is required to behave in a predictable manner; it must

support properties that enable predictions on behaviour and duration of event propagation and delivery. The predictable behaviour of STEAM needs to be enforced by the algorithm that matches events against potential filters, by the mechanism that propagates events on behalf of a producer to the subscribed consumers, and by the scheduler that delivers events to the application. Different scheduling algorithms for delivering events to the application in a deterministic manner have been proposed in [14] and predictable propagation of event messages are addressed in the underlying mechanism for proximity-based group communication [12]. Matching events against a potentially large number of subscriptions, i.e., filters, in a predictable manner is non trivial. Producers exploiting state of the art content-based filters are expected to maintain a very large number of subscriptions and thus need to compare the large number of corresponding filter expressions to the values of an event's parameters. Hence, it will be difficult to implement a matching algorithm that executes in a predictable manner due to the arbitrary number of filters and the arbitrary number of event parameters. In contrast, producers using STEAM's subject-based approach exploit a simple table lookup algorithm to retrieve the list of matching subscriptions, which results in a constant retrieving time. Moreover, the number of the deployed subject filters is expected to be substantially smaller than the number of content filters in a traditional approach as a single filter per subject is required. Our approach of exploiting content filters on the consumer-side faces similar issues as the traditional, producer-side approach. However, we expect our approach to include a substantially smaller number of filters, as we only need to consider the filters that are specific to a single consumer. A sub-linear algorithm suffices to retrieve the filters that correspond to a specific subject using a simple table lookup. Another algorithm is then used to match an event against the set of retrieved filters. We argue that the number of filters at this stage will be very small; there may potentially be a single filter per subject. Computing the match for a single filter is efficient; we propose imposing an upper bound on the computation time for the content filter for each specific subject that can be used to predict the time for matching events of that subject on both the producer and the consumer side.

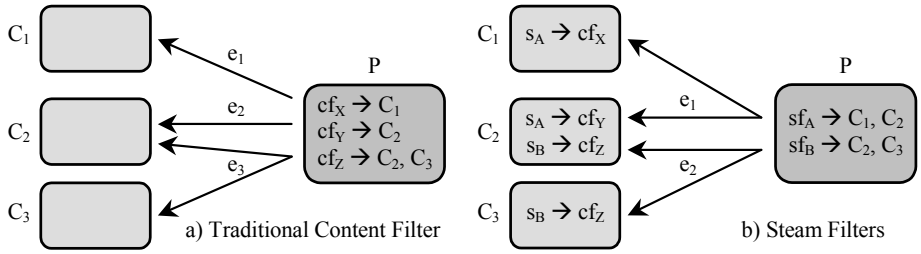


Figure 3. Content vs. STEAM Filters.

Figure 3 compares traditional, content-based filters shown in part a) with STEAM filters outlined in part b). In Figure 3a), a producer *P* matches an event *e* against each of its content filter *cf* to determine the list of consumers C to which to propagate *e*. For example, *e1* matches *cfX* and is therefore propagated and delivered to *C1*. Likewise, *e2* matches *cfY* and is delivered to *C2*. *e3* matches *cfZ*, a filter that has been deployed by both *C2* and *C3*, and is delivered to both *C2* and *C3*. In contrast, *P* in Figure 3b) holds a lookup table containing its subject filters *sf*, each of which relates to one of the types of event that may be produced. For example, *e1* of subject *A* is being propagated to the consumers *C1* and *C2* and *e2* of subject *B* is being propagated to the consumers *C2* and *C3*. Once a consumer receives an event, it uses the event's subject as the key for accessing its lookup table to retrieve the relevant content filter *cf*. e is only delivered to the application if it matches *cf*. *e1* of subject *A* is only delivered to *C1*'s application if it matches *cfX* and to *C2*'s application if it matches *cfY*. In summary, we argue that our approach of deploying subject and content filters on the producer and the consumer side respectively, allows a system to take advantage of the efficiency of subject filters and the expressiveness of content filters while supporting predicable algorithms for matching events and filters. Moreover, the computational load of matching events against filters is distributed between a producer and the subscribed consumers.

4.1.5 Conclusion and Future Work

We have outlined our work on STEAM, an event-based middleware service that has been designed for the mobile computing domain utilizing wireless local area networks and the ad hoc network model. We have outlined and discussed our design regarding the chosen event model and the underlying group communication mechanism and concluded that STEAM must omit application components providing system-wide services. We argue that in the envisaged traffic management application domain entities are more likely to interact once they are in close proximity and propose a novel approach of event filtering by combining three different types of event filter. This approach is well suited for systems that consist of a potentially large number of dynamically joining and leaving entities whose scale may change over time. Although we have addressed some of the fundamental issues arising when applying event-based middleware to the mobile computing domain, certain issues remain open for future work. Failed and temporary unavailable entities and connections are the norm rather that the exception in wireless networks compared to wired environments. A mechanism needs to be provided that anticipates entities and connections that may fail and subsequently handles partitioned proximities.

4.1.6 References

[1] M. Weiser, "Ubiquitous Computing," *IEEE Hot Topics*, vol. 26, pp. 71-72, 1993.

[2] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications," *IEEE Computer*, vol. 33, pp. 68-76, 2000.

[3] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper, "Implementing a Sentient Computing System," *IEEE Computer*, vol. 34, pp. 50-56, 2001.

[4] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS," *IEEE Transaction of Software Engineering (TSE)*, vol. 27, pp. 827-850, 2001.

[5] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine*, pp. 116-126, 1997.

[6] Object Management Group, *CORBAservices: Common Object Services Specification - Notification Service Specification, Version 1.0*: Object Management Group, 2000.

[7] M. Haahr, R. Meier, P. Nixon, V. Cahill, and E. Jul, "Filtering and Scalability in the ECO Distributed Event Model," in *Proceedings of the 5th International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE/ICSE2000)*. Limerick, Ireland: IEEE Computer Society, 2000, pp. 83-95.

[8] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content Based Routing with Elvin4," in *Proceedings of AUUG2K*. Canberra, Australia, 2000.

[9] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," in *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*. Vienna, Austria, 2002.

[10] F. Cristian, "Synchronous and Asynchronous Group Communication," *Communications of the ACM*, vol. 39, 1996.

[11] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A Case for Message Oriented Middleware," presented at 13th International Symposium on DIStributed Computing (DISC'99), Bratislava, Slovak Republic, 1999.

[12] M. O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards Group Communication for Mobile Participants," in *Proceedings of Principles of Mobile Computing (POMC'2001)*. Newport, Rhode Island, USA, 2001, pp. 75-82.

[13] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, pp. 283 - 331, 2001.

[14] T. Harrison, D. Levine, and D. Schmidt, "The Design and Performance of a Real-Time CORBA Event Service," in *Proceedings of the 1997 Conference on Object- Oriented Programming Systems, Languages and Applications (OOPSLA)*. Atlanta, Georgia, USA: ACM Press, 1997, pp. 184-200.

4.2 Taxonomy of Distributed Event-Based Programming Systems

René Meier and Vinny Cahill

Department of Computer Science, Trinity College Dublin, Ireland

{rene.meier, vinny.cahill}@cs.tcd.ie

Abstract - This paper presents a survey of existing event systems structured as a taxonomy of distributed event-based programming systems. Our taxonomy identifies a set of fundamental properties of event-based programming systems and categorizes them according to the event model and event service criteria. The event service is further classified according to its organization and interaction model, as well as other functional and non-functional features.[6]

4.2.1 Introduction

Event-based middleware is currently being applied for application component integration in many application domains including finance, telecommunications, smart environments, multimedia, avionics, health care, and entertainment and event services are omnipresent in applications ranging from small-scale, centralized to large-scale, highly distributed systems. As event-based middleware is exploited in a number of applications in a range of domains, a variety of event services have been proposed to address different application requirements. This paper presents a survey of existing event systems structured as a *taxonomy of distributed event-based programming systems*. Our taxonomy identifies a set of fundamental properties of distributed event-based programming systems, or simply event systems, and categorizes them according to the *event model* and *event service* criteria. The latter is further classified according to its *organization*, *interaction model*, and its *functional* and *non-functional features*. These properties are then arranged in a hierarchical manner starting from the root dimension of the taxonomy, which defines the relationship between event system, event service and event model. Every event system, which we define as an application that uses an event service to carry out event-based communication, has both an event service and an event model. We define an event service as the middleware that implements an event model, hence providing event-based communication to an event system. An event model consists of a set of rules describing a communication model that is based on events. The following sections introduce the event model and the event service dimension of our taxonomy. Figures are presented to outline the relationship among the fundamental properties of event systems and to define the terminology to identify them. Existing event systems are applied to the taxonomy to further outline the identified properties. However, we omit an in dept discussion of our taxonomy and of the given examples due to space limits. A more detailed description and discussion can be found in [1].

4.2.2 Event Model Dimension

The event model defines the application view of an event service. It defines the manner in which an event service is made visible to the application programmer and specifies the components of an event service to which the application programmer is explicitly exposed. Specifically, it classifies the means by which the consuming entities of an application

[6] The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET programme of the Commission of the EU under research contract IST-2000-26031 (CORTEX).

subscribe to the events in which they are interested and the means by which an application raises and delivers events.

We have identified three distinct categories of event model, which are peer to peer, mediator, and implicit. A peer to peer event model allows consuming entities to subscribe at specific named producing entities directly and producing entities to deliver events to specific named subscribed entities directly. For example, the Java distributed event model is based on a peer to peer event model. Event models utilizing a mediator allow consuming entities to subscribe at a designated mediator and producing entities to deliver events to the mediator, which then forwards them to the subscribed consumers. The CORBA event model uses a mediator, called event channel, through which events are propagated. An implicit event model lets consuming entities subscribe to particular event types rather than at another entity or a mediator. Producing entities generate events of some type, which are then delivered to the subscribed consumers. The Cambridge event architecture [2] is based on an implicit event model.

4.2.3 Event Service Dimension

The event service dimension deals with the classification of the properties of an event service, which we divide into three distinct categories. The organization sub tree focuses on the distribution of the entities and the middleware of an event system and on the fashion in which the components that comprise an event service cooperate. The interaction model defines the communication path over which producing and consuming entities communicate with each other. It defines the number of intermediate middleware components involved and the manner in which intermediates cooperate to route events from producers to consumers. The feature sub hierarchy addresses the other functional and non-functional features proposed by an event service.

The organization sub tree classifies an event service as either centralized or distributed according to the location of the event system's entities. The entities are centralized if they only reside in the same address space on the same physical machine. In contrast, if the entities of an event system are distributed they may be located in different address spaces possibly on different physical machines. Figure 1 outlines that these two sub categories are further divided exploring the location of the event service middleware. A distributed organization with collocated middleware has been adopted by mSECO [3], which is exclusively located in the same address spaces as the entities. SIENA [4] proposes a set of middleware topologies of which all but the centralized topology use middleware that is distributed over a set of cooperating machines, thus utilizing a distributed organization with separated middleware.

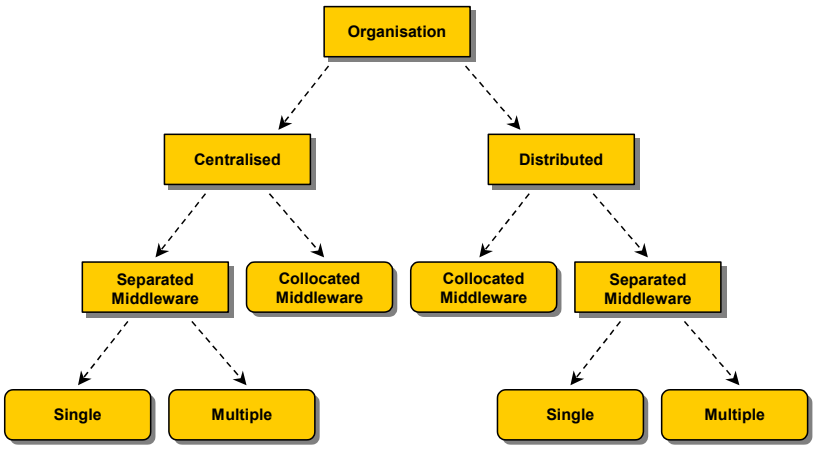


Figure 1. Event Service Organization.

The interaction sub tree classifies an event service according to the interaction model used by the event system. Compared to the organization model, which focuses on the distribution of the entities and the middleware of an event system describing the static view of an event

service, the interaction model describes the information flow in an event system. Hence, it describes the dynamic aspect of an event service. As Figure 2 depicts, we divide the interaction model into two main categories, namely intermediate and no intermediate, exploring whether and how many intermediate middleware components an event passes through. Both categories can be divided further. For example, JEDI [5] proposes a hierarchical structure of cooperative distributed intermediates, called dispatching servers, which are interconnected in a tree topology through which events are routed. In contrast, uSECO [3] does not utilize intermediates but uses a name service to resolve the addresses of the entities to which events are routed.

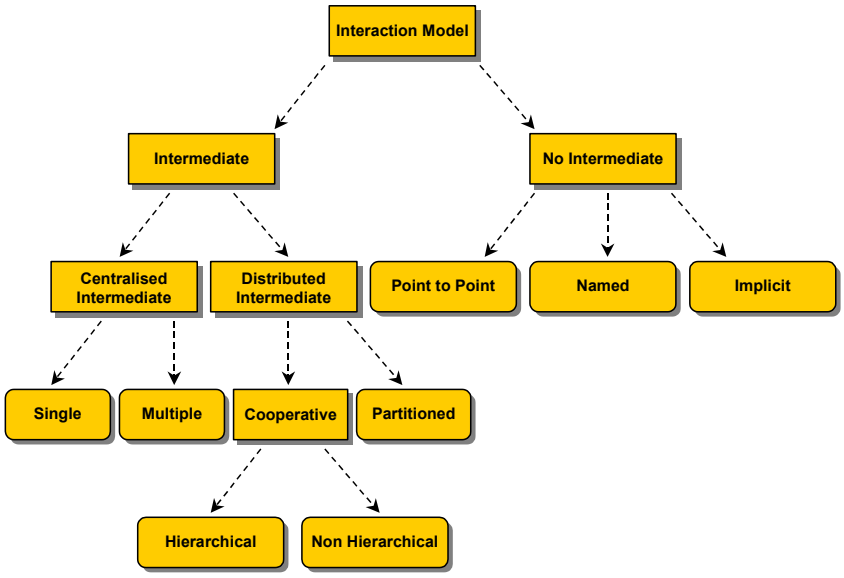


Figure2. Event Service Interaction Model.

The feature sub hierarchy includes functional features such as event propagation model, event type and filter, mobility and composite events, as well as non-functional features such as QoS, ordering, and fault tolerance.

4.2.4 References

[1] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," Dept. of Computer Science, Trinity College Dublin, Ireland, Technical Report TCD-CS-2002, March 2002.

[2] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications," *IEEE Computer*, vol. 33, pp. 68-76, 2000.

[3] M. Haahr, R. Meier, P. Nixon, V. Cahill, and E. Jul, "Filtering and Scalability in the ECO Distributed Event Model," in *Proceedings of the 5th International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE/ICSE2000)*. Limerick, Ireland: IEEE Computer Society, 2000, pp. 83-95.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, pp. 283 - 331, 2001.

[5] G. Cugola, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS," *IEEE Transaction of Software Engineering (TSE)*, vol. 27, pp. 827-850, 2001.

Chapter 5: Quality of Service Specification

This chapter presents two position papers addressing the quality of service specification in the CORTEX programming model. Section 5.1 presents a paper outlining how QoS properties may be specified in the CORTEX programming model and section 5.2 introduces a paper on fundamental issues of dependable and timely computing with a TCB.

5.1 Quality of Service in the CORTEX Programming Model

CORTEX aims to explore the fundamental theoretical and engineering issues necessary to support the use of sentient objects to construct large-scale proactive applications. The programming model supports the development of applications constructed from mobile sentient objects. The model needs to take into account the provision of incremental real-time and reliability guarantees. To achieve this, the development of a means to express quality of service (QoS) properties in the programming model, where QoS is a metric of predictability in terms of timeliness and reliability is required. Additionally a global model for QoS assurance across heterogeneous physical networks is essential. This section will outline how QoS properties may be specified in the programming model.

5.1.1 Introduction

CORTEX applications will be composed of collections of sentient objects. Sentient objects must be able to discover and interact with each other and with the physical world in ways that demand predictable and sometimes guaranteed QoS, encompassing timeliness and reliability guarantees. Achieving predictability is made difficult by the characteristics of the changing environment in which these objects operate, including unstable and mobile object population, unpredictable network load, varying connectivity, and the presence of failed system components. Thus, the construction of applications from sentient objects must take into account the fundamental trade-off between the existence of a dynamic environment and the need for predictable operation.

In the CORTEX domain, the programming model supports the development of applications constructed from mobile sentient objects. The model needs to take into account the provision of incremental real-time and reliability guarantees. How will the programming model support these requirements in a heterogeneous environment of varying performance capabilities? The programming model must enable the application developer to specify high-level application level QoS parameters and to assure QoS guarantees irrespective of the heterogeneity of the networks. How will the programming model fulfill these requirements whilst retaining a complete abstraction from the low-level system properties? To answer these questions, an open, scalable, system architecture, that reflects the heterogeneity and performance of the networks used to support the programming model, must be defined. The programming model must be presented with an abstraction of the architecture and temporal properties of the internetwork, sufficiently robust to satisfy the QoS requirements stated. The rest of this paper will investigate the abstraction interface and QoS assurances available to the programming model.

5.1.2 QoS Interfaces for the Programming Model

5.1.2.1 Abstract Network Model

The CORTEX programming model must facilitate the non-functional requirement of enabling timeliness and reliability properties related to the delivery of event notification. The mechanism provided for QoS specification is based on the use of application-level QoS parameters that can be mapped onto system-level QoS parameters characterizing the service levels that can be supported by the underlying physical infrastructure at a given point in time. This mechanism enables application developers to specify QoS requirements in terms that are

meaningful for particular application areas, while also supporting the degree of abstraction necessary to accommodate variations in the QoS that can be supported by the underlying environment.

To enable QoS specification, the CORTEX programming model will make the heterogeneity of the underlying physical system visible, at a high level of abstraction. To facilitate this, a complete abstract network model of all physical networks comprising the internetwork will be available to the programming model. Thus the environment may be modeled as an abstract network at the programming model level, enabling the application developer to specify application level QoS constraints. How are these application QoS requirements reflected at the system level, where the underlying heterogeneity of the environment is exposed?

The CORTEX programming model relies on an anonymous event-based communication model for the propagation and delivery of events. The event model will provide mechanisms to constrain event notification by providing incremental real-time and reliability guarantees. This will encompass developing a suitable representation for expressing QoS properties, which will be presented to the application developer.

In CORTEX, complete mappings from the application levels', abstract network model, to the corresponding physical internetwork's, timeliness and reliability properties will be available. Thus, an application developer specifies QoS requirements at the application context, which are mapped to specific network QoS properties at the system level.

5.1.2.2 Zones

The basis for the proposed CORTEX architecture is to model the underlying communication infrastructure hierarchically, structured as a WAN-of-CANs. If we take a typical requirement from the ATTS scenario of [2]. Vehicles communicate to provide a look-ahead warning service for vehicles coming from behind. If a vehicle detects an obstacle it sends an alert, which the receiving participants can exploit to set new cruising parameters, and further disseminate the message. Problems associated with this scenario are discussed in [1], and relate to the problems of co-operation and communication between vehicles. Additional specific problems relating to the hierarchy of communication networks present inside a vehicle, to interpret and act on the event notification must also be considered.

The network topology is viewed as an internetwork whose subnetworks are typically CANs, which are interconnected by means of LANs and WANs. The QoS available for the internetwork varies per network. For example, a CAN has strong timing guarantees, with LANs and WANs providing weaker timing guarantees. Individual networks can be viewed as guaranteeing a given QoS degree, and can be viewed as *QoS containers*.

CORTEX supports the construction of advanced proactive applications, comprising mobile sentient objects with autonomous behavior resulting from interactions with other objects and with the physical environment. Dependability when confronted with unpredictable interaction patterns, must be available, and is achieved through QoS parameter specification, as derived in the application domain. Co-operating communities of sentient objects can thereby interact to achieve zones of QoS coverage where predictable behavior is ensured and graceful degradation and failure modes are possible.

As stated previously, the heterogeneity of the underlying physical system must be made visible to the application developer at an abstract level. To take advantage of the varying timing guarantees, dependent upon the specific network in use, the QoS containers will be visible as a hierarchy of zones, capable of delivering specified levels of QoS. With respect to the vehicle example above, there are three different zones: the vehicle (internal zone), the immediate proximity of the vehicle (the external zone) and remote proximity (distant zones). Each of these zones is an island of control, with specific QoS guarantees. In general terms, each of these zones must co-operate via gateways in a timely and reliable manner.

A zone identifies a natural border for the propagation of broadcast messages at specific quality attributes, like certain reliability of message transfer or guaranteed bounds on

transmission time. The notion of a group is orthogonal to a zone. Groups are logical entities, with a common interest; some examples are proximity or subject. Thus, groups may be completely contained within the one zone or group members may reside in different zones. The latter case is of particular interest and will be investigated further.

An important issue for group co-operation is to know what QoS can be sustained by the zones in which group members reside. A CAN represents a zone with a high level of predictability, in comparison to a wireless network zone, where QoS guarantees are necessarily much weaker.

In a mobile environment, migration from one zone to another is likely to happen. Group members may be widely spread over many zones, subsequently having differing QoS levels. If group members reside in different zones, communication within the group will adapt to the weakest zone guarantees. It is essential to provide highly adaptive behavior accounting for dynamically varying network parameters. In addition, there must be timeliness predictability, even if the system is degrading, it should be done in a predictable manner. The coverage of timeliness assumptions should remain stable throughout the applications lifetime. At the system architecture level, *gateways* aimed at hiding the differences implicit in the various physical networks (like addressing structures) and the quality attributes available to each, will be developed.

Gateways provide a representation of a certain environment to the outside world, essentially to the programming model. At a logical level, the programming model will use the environment abstraction that defines zones of similar QoS guarantees with gateways lying in the border of these zones, to specify and observe QoS guarantees. Thus, at the programming model level the heterogeneous networks may be viewed as a hierarchy of zones, capable of delivering specified levels of QoS. An application developer may specify application level QoS guarantees for the zones presented, which will be mapped to system level timeliness and reliability properties.

5.1.3 Summary

In the CORTEX domain, the programming model supports the development of applications constructed from mobile sentient objects. The model needs to take into account the provision of incremental real-time and reliability guarantees. The application developer must remain oblivious of the heterogeneity of the physical networks involved. This implies that the programming model must be presented with an abstraction of this architecture, sufficiently robust to satisfy the QoS requirements stated.

The mechanism provided for QoS specification is based on the use of application-level QoS parameters that can be mapped onto system-level QoS parameters characterizing the service levels that can be supported by the underlying physical infrastructure at a given point in time.

There is a natural QoS hierarchy in CORTEX, which has varying QoS guarantees dependent upon the current network, and contained within that network. These QoS containers will be visible as a hierarchy of zones, capable of delivering specified levels of QoS.

Thus, at the programming model level the heterogeneous networks may be viewed as a hierarchy of zones, capable of delivering specified levels of QoS. An application developer may specify application level QoS guarantees for the zones presented, which will be mapped to system level timeliness and reliability properties.

5.1.4 References

[1] CORTEX Annex1 – “Description of Work”, October 2000.

[2] CORTEX Definition of Application Scenarios, v1.0, October 2001.

5.2 Dependable and Timely Computing with a TCB: Fundamental Issues

Quality of Service can have different meanings that depend essentially on the application. This is why the definition of a completely generic model for specifying QoS needs is perhaps an impossible task: it is usually necessary to use mapping mechanisms to translate user level QoS requirements into system level ones. It is usually necessary to assume that it is possible to define mapping mechanisms from application to system level, so that QoS requirements of different applications can be specified in terms of timing variables, which are the variables of interest in the TCB model.

The TCB model provides a generic framework to deal with synchronism problems and to provide certain safety and liveness properties in the time domain to applications. In this sense, there is a potential for this model to be used as a base model for the development of some application classes, namely for the class of adaptable applications with QoS requirements.

Here we investigate and discuss the basic issues that are necessary to allow the specification and the design of applications with QoS requirements, as well as the definition of adequate programming interfaces that take into account these QoS requirements. This is obviously done in the framework of the TCB model, which we briefly describe in the next section. A more detailed description of the TCB model and its impact on the CORTEX architecture is presented in Deliverable WP3-D4. Since the possibility of guaranteeing a certain QoS to applications depends on the ability of monitoring capabilities, which in fact can be viewed as a form of context and environment awareness, we present in Deliverable WP2-D3 more details on dealing with QoS issues under the TCB model, addressing in particular the problem of dependable QoS adaptation.

5.2.1 Basic Description of the TCB Model

A system with a TCB is divided into two well-defined parts: a *payload* and a *control* part. The generic or *payload* part prefigures what is normally 'the system' in homogeneous architectures. It exists over a global network or *payload* channel and is where applications run and communicate. The *control* part is made of local TCB modules, interconnected by some form of medium, the *control* channel. Processes *p* execute on several sites, making use of the TCB whenever appropriate. Figure 1 illustrates the architecture of a system with a TCB.

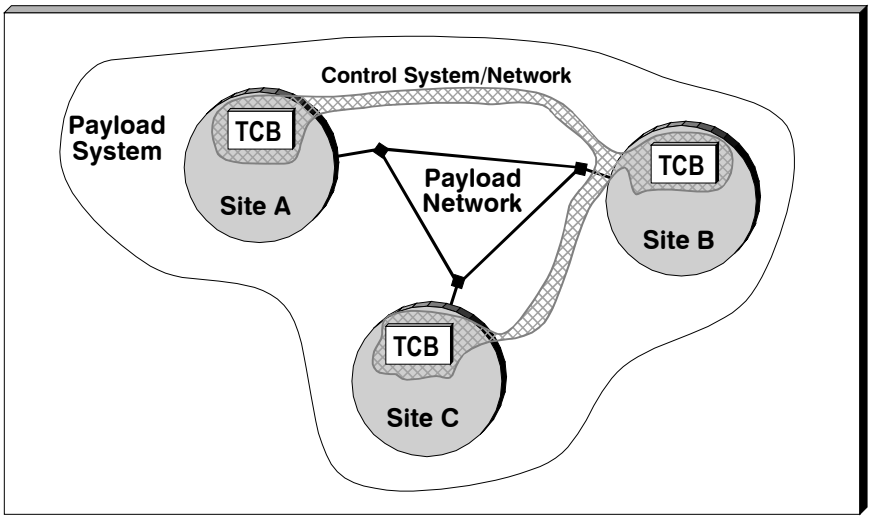


Figure 1: The TCB Architecture

Concerning the payload part, the important property is that the system *can have any degree of synchronism*, that is, if bounds exist for processing or communication delays, their magnitude may be uncertain or not known. Local clocks may not exist or may not have a bounded rate of drift towards real time. The system is assumed to follow an omissive failure model, that is, components *only do timing failures*– and of course, omission and crash, since they are subsets of timing failures– no value failures occur.

In the control part, there is one local TCB at every site, fulfilling the following construction principles:

Interposition - the TCB position is such that no direct access to resources vital to timeliness can be made in default of the TCB;

Shielding - the TCB construction is such that it itself is protected from faults affecting timeliness;

Validation - the TCB functionality is such that it allows the implementation of verifiable mechanisms w.r.t. timeliness.

TCB modules are assumed to be fail-silent, that is, they only fail by crashing. Moreover, it is assumed that the failure of a local TCB module implies the failure of that site. In terms of synchrony, the TCB subsystem preserves, by construction, upper bounds on processing delays, on the drift rate of local TCB clocks and on the delivery delay of messages exchanged between local TCBs.

Given the above set of construction principles and properties, a TCB can be turned into an oracle for applications (even asynchronous) to solve their time related problems. To accomplish this, a set of minimal services has to be defined, as well as the payload-to-TCB interface.

5.2.1.1 TCB Services

In order to keep the TCB simple, the services defined are only those essential to satisfy a wide range of applications with timeliness requirements: ability to measure distributed durations with bounded accuracy (provided by the **Duration Measurement** Service); complete and accurate detection of timing failures (provided by the **Timing Failure Detection** Service); ability to execute well-defined functions in bounded time (provided by the **Timely Execution** Service).

5.2.1.2 Providing Adequate Programming Interfaces

Besides defining essential services to be provided by the TCB, it is very important to design a programming interface to allow potentially asynchronous applications to dialogue with a synchronous component. A relevant aspect to understand what can be done is that applications can only be as timely as allowed by the synchronism of the payload system. The TCB, although being a synchronous component does not make applications timelier, it only provides the means to detect how timely they are. Another important aspect is that nothing obliges an application to correctly use, or use at all, the TCB capabilities. Applications are *autonomous entities* that take advantage of the TCB by construction. They typically use it as a pacemaker, letting it assess (explicitly or implicitly) the correctness of past steps before proceeding to the next step. Translating this behavior into a QoS approach, this means that applications requiring a certain QoS level must be constructed in such a way that they use and exploit the capabilities of the TCB.

5.2.2 Effect of Timing Failures

Since we are considering that QoS issues must be addressed in the context of the TCB, our reasoning must be done in terms of timeliness, and in terms of the fundamental obstacle to timeliness, which is the occurrence of timing failures. Therefore, when the objective is to achieve dependable and timely computing, and in particular to deliver a certain QoS, the question to be asked is *How can the TCB assist applications in the presence of failures?* A

constructive approach consists in analyzing why systems fail in the presence of uncertain timeliness, and deriving sufficient conditions to solve the problems encountered, based on the behavior of applications and on the properties of the TCB.

We use 'application' to denote a computation in general, defined by a set of safety and timeliness properties, $\mathcal{P}_A$. We note that in the considered TCB model components only do late timing failures. In the absence of timing failures, the system executes correctly. When timing failures occur, there are essentially three kinds of problems, which we define and discuss below: *unexpected delay*; *contamination*; and *decreased coverage*.

The immediate effect of timing failures may be twofold: unexpected delay and/or incorrectness by contamination. We define **unexpected delay** as the violation of a timeliness property. That can sometimes be accepted, if applications are prepared to work correctly under increased delay expectations, or it can be masked by using timing fault tolerance. However, there can also be **contamination**, which we define as the incorrect behavior resulting from the violation of safety properties on account of the occurrence of timing failures. This effect has not been well understood, and haunts many designs, even those supposedly asynchronous, but where aggressive timeouts and failure detection are embedded in the protocols[1,2]. In fact, problems such as described in [1,3] assume a simple dimension, when explained under the light of timing failures. These designs fail because: (a) although time-free by specification, they rely on time, often in the form of timeouts; (b) they are thus prone to timing failures (e.g., when timeouts are too short); (c) however, proper measures are not taken to counter timing failures (because they were not supposed to exist in the first place!); (d) in consequence, error confinement is not ensured, and sometimes timing failures contaminate safety properties[7]. A particular form of the contamination problem was described in the context of ordered broadcast protocols in [4].

In this paper, we give a generic definition of the problem, for a system with omissive failures. If the system has the capacity of *timely detecting timeliness violations*, then contamination can be avoided with adequate algorithm structure. To provide an intuition on this, suppose that the system does not make progress without having an assurance that a previous set of steps was timely. Now observe that "timely" means that the detection latency is bounded. In consequence, if it waits more than the detection latency, absence of failure indication means "no failure", and thus it can proceed. If an indication does come, then it can be processed before the failure contaminates the system, since the computation has not proceeded. The only consequence of this mechanism is an extra wait of the order of the detection delay (which is bounded, according to the properties of TCB services).

A sufficient condition for absence of contamination is expressed by the *No-Contamination* property. Informally, this condition stipulates that upon occurrence of a timing failure, its effect is confined to the violation of timeliness properties alone.

The reader will note that in the model of Chandra[2], the agreement algorithms have no-contamination, since their design is completely time-free, and all possible problems deriving from timing failures (such as "wrong suspicions") are encapsulated in the failure detector. Chandra then bases the reliability of his system on the possibility of implementing a given failure detector, but he does not discuss this implementation. In contrast, in the TCB framework there exists a placeholder for the viable implementation of special services such as failure detection– the TCB control part, aside of the TCB payload part, containing the algorithms or applications. One important advantage is generality of the programming model, by letting the payload system have any degree of synchrony. That is, we devise a single framework for correct execution of synchronous and asynchronous applications, of several grades that have been represented by partial models such as asynchronous with failure

[7] As a matter of fact they may also contaminate liveness properties, by preventing progress. We do not explicitly discuss liveness properties in this paper.

detectors, timed asynchronous, or quasi-synchronous. Or, in other words, from non real-time, through soft, mission-critical, to hard real-time.

Let us talk now about **decreased coverage** as the other effect of timing failures. Whenever we design a system under the assumption of the absence (i.e., prevention) of timing failures, we have in mind a certain coverage, which is the degree of correspondence between system timeliness assumptions and what the environment can guarantee. We define *assumed coverage* P_P of a property P as the assumed probability of the property holding over an interval of reference. This coverage is necessarily very high for timeliness properties of hard real-time systems, and may be somewhat relaxed for other realistic real-time systems, like mission-critical or soft real-time. Now, in a system with uncertain timeliness, the abovementioned correspondence is not constant, it varies during system life. If the environment conditions start degrading to states worse than assumed, the coverage incrementally decreases, and thus the probability of timing failure increases. If on the contrary, coverage is better than assumed, we are not taking full advantage from what the environment gives. Both situations are undesirable, and this is a generic problem for any class of system relying on the assumption/coverage binomial [5]: if coverage of failure mode assumptions does not stay stable, a fault-tolerant design based on those assumptions will be impaired. A sufficient condition for that not to happen consists in ensuring that coverage stays close to the assumed value, over an interval of mission.

This is expressed by what we have called the *Coverage Stability* property, which can be informally explained as follows. If we adapt our timeliness requirements say, by relaxing them when the environment is giving poorer service quality, and tightening them when the opposite happens, we maintain the actual coverage of the implementation in runtime around a desirably small interval of confidence, p_{dev}, of the assumed coverage (P_P). The interval of confidence is the measure in which coverage stability is ensured. Note that even if long-term coverage stability is ensured, instantaneously the system can still have timing failures, as we have previously discussed.

5.2.3 A QoS Model

Timing specifications, the ones that are typically interesting in the context of the TCB, are usually derived from application timeliness requirements. Since the kinds of environments foreseen in CORTEX do not allow to assume a guaranteed behavior, time bounds stipulated by applications may be violated during execution. Therefore, instead of assuming and reasoning exclusively in terms of bounds, a better approach is to consider that time bounds have an associated measure of the probability that they will hold during an interval of execution. This measure corresponds, in fact, to the **coverage** of the assumption that the time bound will hold, as we have discussed above.

The possible consequences for the definition of an adequate programming model, which takes into account QoS requirements, is that the latter should be specified through <bound, coverage> pairs, that is, by defining a bound that should be secured with a given coverage. The development of dependable applications with respect to their QoS requirements must ensure that QoS specifications of the form <bound, coverage> will be secured. Incidentally, we should point out that this approach might be applied in soft, as well as in mission-critical real-time systems design. Timeliness of execution is guaranteed with a certain coverage (the QoS specification) and, should a QoS failure be detected, safety measures (e.g., fail-safe shutdown), or real-time adaptation (e.g., reducing the system requirement), or QoS renegotiation can be undertaken, depending on the application characteristics [6].

Showing that the TCB can help applications to secure their QoS specifications (which means, in the TCB model, securing coverage stability and no-contamination), despite the uncertainty of the environment, is done in deliverable WP2-D3, in the context of the CORTEX interaction model. We propose three classes of applications, which can be combined in the solution of concrete problems. The *time-elastic* class is oriented to securing coverage stability under a

varying environment, for example achieving what we have called *dependable QoS adaptation* (see WP2-D3). The *fail-safe* class provides guidelines for ensuring the safe shutdown upon a timing failure that cannot be handled, and before contamination occurs (see WP2-D3 for examples). The *time-safe* class also aims at guaranteeing no-contamination, but this time by providing conditions for operation to continue. In all that follows, we consider the availability of the TCB services.

5.2.4 References

[1] T. Chandra and V. Hadzilacos and S. Toueg. On the Impossibility of Group Membership. *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pp.322-330. Philadelphia, USA. May 1996.

[2] T. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2):225-267. March 1996.

[3] E. Anceaume and B. Charron-Bost and P. Minet and S. Toueg. On the Formal Specification of Group Membership Services. Technical Report RR-2695, INRIA, France, November 1995.

[4] A. Gopal and S. Toueg. Inconsistency and Contamination. *Proceedings of the 10th Annual {ACM} Symposium on Principles of Distributed Computing*, pp.257-272. Montreal, Québec, Canada. 1991.

[5] D. Powell. Failure Mode Assumptions and Assumption Coverage. *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*. pp.386-395. Boston, USA. July 1992.

[6] P. Veríssimo and L. Rodrigues. Distributed Systems for System Architects. Kluwer Academic Publishers. 2001.

Chapter 6: References

[1] CORTEX Technical Annex 1, Description of Work, October 2000.

[2] CORTEX Deliverable D1, Definition of Application Scenarios, October 2001.

[3] H. Kopetz, M. Holzmann, W. Elmenreich:"A Universal Smart Transducer Interface: TTP/A", *Int. Journal of Computer System Science & Engineering,* 16(2), March 2001.

[4] Object Management Group: "Smart Transducer Interface", Request for Proposal, OMG Document: orbos/2000-12-13, Dec. 2000.

[5] Alireza Moini: Vision Chips or Seeing Silicon, Third Revision, http://www.eleceng.adelaide.edu.au/Groups/GAAS/Bugeye/visionchips/index.html, March 1997.

[6] J. Kaiser, P. Schaeffer: " ICU – A smart optical sensor for direct robot control", *Proc. IEEE International Conference on Mechatronics and Machine Vision in Practice 2001,* Hong Kong, China, August 27-29, 2001.

[7] International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria, July 2002.